UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Modulo de creación de laberintos en Unity.

Raquel Montero Sacristán
Tutor: Carlos Aguirre Maeso

Junio 2016

Modulo de creación de laberintos en Unity.

AUTOR: Raquel Montero Sacristán
TUTOR: Carlos Aguirre Maeso

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2016

Resumen (castellano)

Mediante el motor gráfico de Unity se pueden crear laberintos estáticos, predefinidos antes del tiempo de ejecución, como planos con mucha facilidad.

El presente trabajo se basa en poder proporcionar la misma facilidad para un usuario de crear laberintos aleatorios como la que se proporciona por el entorno de desarrollo mediante interfaz.

Para ello se han creado unos scripts en donde se encapsulan las necesidades más imprescindibles que puede tener un usuario a la hora de crear un laberinto aparte de crear un laberinto, como es el algoritmo a usar, el tamaño del mapa, el ajustamiento del terreno al laberinto, la inserción de elementos dentro del laberinto, eliminar el laberinto, añadir navegación dentro del laberinto junto a todos los objetos que están parametrizados como elementos de navegación.

El laberinto inicialmente es un GridLayout el cual posee una estructura Celda la cual posee las coordenadas de los muros norte, sur, este y oeste, sus instanciaciones un indicador de si ha sido visitado o no y un indicador de si el muro es insertable o no.

Los algoritmos de generación de laberintos son varios, pensados para distintas necesidades de generación, cada cual tiene sus pros y sus contras:

- **Algoritmo DFS**: también conocido como búsqueda primero en profundidad, el cual genera un laberinto tradicional perfecto en donde no existen zonas cerradas a las cuales no se pueda acceder y no posee bucles por lo que si se recorriese siempre el lado izquierdo o derecho del laberinto se alcanzarían todas las zonas sin importar la forma del laberinto.

- **Algoritmo Árbol binario**: este algoritmo se basa en recorrer cada una de las celdas en las que está dividido el laberinto y aleatoriamente para cada una de las celdas indica cuales son los muros que no pertenecen al laberinto. El laberinto que se genera tiene ciertas características entre la que destaca su generación en forma escalonada.

- **Algoritmo Árbol binario imperfecto**: similar al árbol binario pero dejando zonas cerradas para poder generar laberintos de cuarta dimensión mediante la inclusión de portales.

- **Algoritmo DFS con estados repetidos:** basado en el algoritmo de búsqueda en profundidad el cual cuando alcanza una celda ya repetida, mediante un ratio de

probabilidad, la cual es modificable por el usuario, se le indicaría al azar cual de sus muros no sería insertado.

Aparte del Asset del gestor de laberintos, se han implementado algunas aplicaciones para comprobar el funcionamiento del Asset, entre las que destaca un juego del Pacman en donde se muestra la utilización del Asset para generar el mapa del Pacman y que en cada ejecución sea uno distinto, en este juego se utiliza tanto el indicador de tamaño, la probabilidad de generar estado repetido, la navegación y la inserción de elementos dentro del laberinto.

Abstract (English)

Using the Unity graphics engine you can create static mazes, before runtime predefined as planes very easily.

This work is based on being able to provide the same facility for a user to create random mazes as that provided by the development environment through interface.

This would have created some scripts where the most essential needs that can have a user when creating a maze apart from creating a maze, as is the algorithm used, the size of the map, the adjustment of the land are encapsulated to maze, inserting elements within the maze, eliminating the maze, add navigation within the maze with all objects that are parameterized as navigation elements.

The labyrinth initially is a GridLayout which has a cell structure which has the coordinates of the north walls, south, east and west, its instantiations an indicator of whether it has been visited or not and an indicator of whether the wall is insert or not .

The maze generation algorithms with several, designed for different needs of generation, each has its pros and cons:

- **DFS Algorithm:** also known as depth-first search, which generates a perfect traditional labyrinth where there are no closed areas which cannot be accessed and has no loops so if the left or right of the labyrinth side always traveler all areas regardless of the form of the maze would be achieved.

- **Binary Tree Algorithm:** This algorithm is based on tour each of the cells in which the maze is divided and randomly for each of the cells indicates what the walls outside the maze. The maze is generated has certain characteristics including notably his generation in stages.

- **Imperfect binary tree algorithm:** binary tree similar to but leaving closed to generate labyrinths fourth dimension by including portal areas.

- **DFS algorithm with repeated states:** based on the search algorithm in depth which when it reaches an already repeated cell, using a likelihood ratio, which is modifiable by the user, will indicate random which its walls would not be inserted.

Apart from the Asset Manager mazes, have implemented some applications to check the operation of the Asset, among which a game of Pacman where the use of Asset shown to

generate the map Pacman and that each execution is a different one in this game both used the size indicator, the probability of generating been repeated, navigation and insertion of elements within the maze.

Palabras clave (castellano)

Motor gráfico, Unity, laberinto, GridLayout, Celda, Algoritmo DFS, Algoritmo Árbol binario, Algoritmo Árbol binario imperfecto, Algoritmo DFS con estados repetidos, Asset, Pacman.

Keywords (inglés)

graphics engine, Unity, labyrinth, maze, GridLayout, Cell, DFS Algorithm, Binary Tree Algorithm, Imperfect binary tree algorithm, DFS algorithm with repeated states, Asset, Pacman.

Agradecimientos

Los agradecimientos merecen ser recordados no solo por la gente que ha estado contigo en los últimos meses durante la realización de el presente trabajo, sino por todas esas personas que me han acompañado durante todo el camino ya sea desde que naci o desde que entre en la carrera.

Existen personas que por algún u otro motivo ya no están tan presentes en mi vida, viejas amistades, amigos recientes, sin embargo, todos en su conjunto en mayor o menor medida han hecho que sea quien soy hoy y que haya decidido seguir el camino de la informática. Por ello deseo agradecer abiertamente a los siguientes grupos de personas:

A mi familia, la cual ha estado ahí, ayudando a levantarme cuando me caía y en momentos en los que no he podido seguir adelante. Gracias por aguantarme todos estos años.

Gracias a mis amigos que saben lo liada que siempre estoy y no me toman en cuenta el hecho de que muchas veces parezca que se me ha tragado la tierra, son gente realmente comprensiva.

Gracias a Manu por todos estos años pues me ha animado mucho a seguir adelante.

Gracias a Keko por ser un gran apoyo durante la carrera, siempre animándome y haciéndome reír.

Gracias a los profesores que he tenido pues no tengo queja de ninguno y en especial a Pablo Varona quien me apoyo bastante mientras estuve en tercero, también a Carlos Aguirre mi tutor de TFG que me ha ayudado bastante con varias de mis asignaturas y con el presente TFG.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

El motor Unity 3D es una herramienta diseñada principalmente para el desarrollo de videojuegos en distintas plataformas, tanto en 2D como 3D.

Este motor ofrece varias alternativas para realizar estas acciones, desde ir insertando los objetos manualmente en un terreno a través de una interfaz gráfica, similar a cuando alguien hace líneas en un Paint y otra alternativa común es la de la inserción de objetos y acciones mediante scripts, ya sean en javascript o en c#.

Lo malo que posee este motor es que hay un gran abismo entre el uso de la interfaz gráfica y el uso de scripts. Si bien es más fácil de utilizar la interfaz gráfica para insertar elementos hacer animaciones, modificar objetos y sus atributos, hacer mapeado, etc. las posibilidades con los scripts a la hora de realizar eventos, modificar objetos y realizar inserciones aleatorias de elementos dentro de unos rangos dados son mucho mayores. La dificultad que tiene este método de script es que Unity está pensado para hacer desde el entorno gráfico muchas de las acciones, por lo que existen ciertos procesos a los cuales no se tienen acceso mediante los scripts y hay que ir con cuidado.

Debido a estas características que posee el motor Unity, se ha considerado que aunque es fácil diseñar un laberinto mediante el entorno gráfico, e incluso Unity ofrece una opción en la que si insertas una imagen de un laberinto en 2D obtenida de cualquier generador de laberintos, como puede ser: http://laberintos.onlinegratis.tv/generador-laberinto-para-imprimir/, aunque los muros que inserta son unos predefinidos por el propio Unity, pero se puede modificar todo lo que ha generado. Un ejemplo de este método mediante una imagen se puede ver aquí: https://www.youtube.com/watch?v=4CN0PObTSHE

Este método de creación de laberintos es adecuado cuando el juego o la aplicación que se desea crear se basara en zonas estáticas y no se desea dar al usuario una sensación de aleatoriedad, pues sino se deberían crear una alta cantidad de laberintos, lo cual a la larga se traduce en un consumo de espacio en disco innecesario, el cual pueda ser necesitado para otras cosas más importantes.

Si bien es cierto que existen ya Assets que tienen la funcionalidad de crear un laberinto aleatorio a partir de unos datos configurables por el usuario y se pueden descargar e instalar desde la propia plataforma Unity, se ha encontrado que a veces sería deseable unos laberintos con ciertas características especiales, por ejemplo, que puedan existir bucles en los caminos, funciones para insertar objetos u obtener posiciones dentro del laberinto, los cuales no están o está solo alguna de las características.

Para solventar estos problemas este Trabajo de Fin de Grado se basa en la creación de estos laberintos a partir de unos parámetros por defecto, configurables por el usuario.

En el script de la creación de laberintos existen varios algoritmos diferentes para la generación del laberinto, para así poder dar al usuario la elección del tipo de generaciones que desea realizar las cuales se adecuen con mayor eficiencia a su aplicación.

1.2 Objetivos

El objetivo principal del TFG es la creación de una librería para la plataforma de Unity, con la característica de que sea un gestor de laberintos, sin especificar en cómo deban ser estos creados, sus características principales ni nada, solo que puedan ser implementados y demostrar la implementación en una aplicación.

Debido a estas características a lo largo de las siguientes paginas se van a ir describiendo cuales son las características de la librería, por que se han seleccionado ciertas metodologías y no otras para la generación, se describirá el hardware utilizado el cual es importante ya que explica el por qué se han tomado ciertas decisiones y no otras y los requisitos mínimos que requiere para nuestro sistema.

Por otro lado, se realizará un análisis de la aplicación que implementa la librería, cómo funciona la implementación y demás características de la misma.

1.3 Organización de la memoria

La memoria consta de 7 capítulos y 4 anexos, los cuales son descritos con más detalle a continuación:

- **Capítulos:** En ellos se recoge todo lo relacionado con la teoría de los laberintos y el gestor de laberintos hasta el capitulo 5 y en el capítulo 6 está dedicado a las implementaciones del gestor en aplicaciones para comprobar su correcta funcionalidad. Se indica una breve descripción de cada apartado a continuación:

    - **1 Introducción:** Es el capítulo en el que se encuentra esta sección. En él se recoge una introducción del TFG, objetivos del TFG y de la memoria y la organización de la memoria con un breve resumen de cada apartado.

    - **2 Laberintos:** Capítulo que no pertenece a la rama de la informática, pero necesario para poder entender el contexto del TFG, en el se habla de qué es un laberinto, tipos de laberintos que se pueden encontrar, un poco de historia para situar en el contexto y entender de donde vienen y su evolución y finalmente algunas de sus aplicaciones actuales, para ver que no es algo solo del pasado sino que es muy utilizado en la actualidad.

    - **3 Estado del Arte:** En este capítulo se describe la tecnología de los laberintos y algunos de los algoritmos más comunes.

    - **4 Análisis y Diseño:** En este capítulo se realiza un análisis de los requisitos tanto funcionales como no funcionales del Asset. También se hace una descripción del diseño utilizado indicando los patrones y el diagrama de clases con una descripción de las clases creadas.

    - **5 Desarrollo:** En este capítulo se cómo se han creado ciertas características del Asset y porqué se ha decidido hacer de cierta forma y no de otra.

- **6 Integración, pruebas y resultados:** En este capítulo se hablará de las aplicaciones en las cuáles se ha insertado el Asset del gestor de laberintos y se indicaran para cada tipo de integración que pruebas se buscaba realizar y cuáles fueron los resultados.

- **7 Conclusiones y Trabajo futuro:** En este capítulo se habla las conclusiones a las qué se ha llegado mientras se realizaba en trabajo y aspectos o características me gustaría añadir al Asset.

2 Laberintos

En este apartado se describirá qué es un laberinto y se hará un breve recorrido de los laberintos a lo largo de la historia, tipos de laberintos y aplicaciones actuales de los mismos para así mas adelante podernos centrar en algoritmos para poder desarrollar este tipo de estructura.

2.1 ¿Qué es un laberinto?

Un laberinto es según una de las definiciones dadas por el diccionario de la lengua española [26] "Lugar formado artificiosamente por calles y encrucijadas, para confundir a quien se adentre en él, de modo que no pueda acertar con la salida". En la **¡Error! No se encuentra el origen de la referencia.** se puede ver un ejemplo de laberinto.

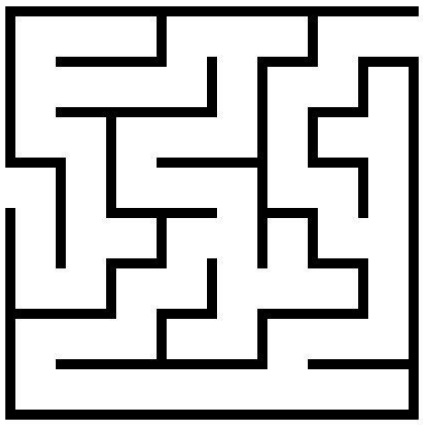


Figura 1: Ejemplo de un laberinto rectangular

El concepto de laberinto viene existiendo desde la antigüedad, en donde ya se relatan historias griegas y romanas acerca de estas estructuras, como es el caso del laberinto del minotauro, en el cual relatan que se construyó de tal manera que era imposible salir y para lograrlo el protagonista se ayudo por un hilo que sujetaban desde fuera.

A lo largo de los siglos los laberintos han ido adquiriendo diversas formas y en la actualidad existen gran diversidad de los mismos.

2.2 Diseñar un Laberinto

Para diseñar un laberinto similar al tipo de la Figura 1, ya sea con un ordenador o en un folio, lo primero es generar una malla, GridLayout o rejilla del tamaño que se desee dar al

laberinto, con tantas columnas y filas como ancho y largo se desee, tal y cómo se ve en la Figura 2.

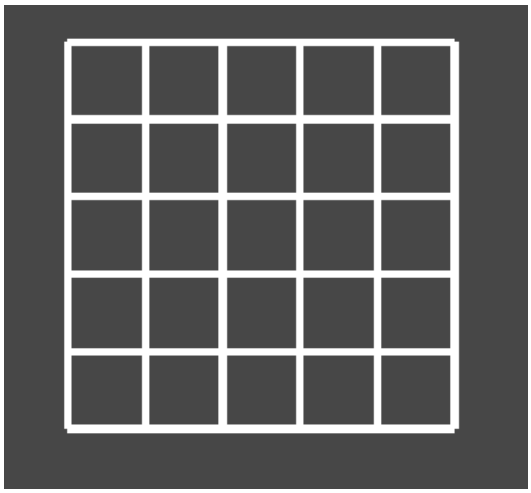


Figura 2: Grid

Cada uno de los cuadrados es lo que se conoce como cell o celda y cada celda posee 4 muros del laberinto que son las 4 orientaciones posibles: norte, sur, este y oeste.

Cada muro pertenece a 1 o 2 celdas, por lo que al borrarlo o se generaría la salida/entrada del laberinto o comunicaría 2 celdas adyacentes, generando un pequeño camino.

A medida que se van eliminando muros, se van generando caminos y recovecos hasta quedar como la Figura 1.

Otra forma es a partir de un folio en blanco o un programa vacío ir insertando los muros.

Para otros tipos de laberinto dependerá de la forma de la malla y del número de muros que posee la celda, pero por lo demás sería muy similar salvo para el tipo circular.

2.3 Tipos de Laberinto.

Los laberintos se pueden clasificar de diversas formas, pero para hacer una rápida indicación de los tipos que se pueden hallar, se puede decir que los laberintos se dividen inicialmente en dos tipos, laberintos de trazado unicursal y laberintos de trazado multicursal, la diferencia entre uno y otro estriba entre que el primero sería un labyrinth, mientras que el multicursal es de categoría maze:

Laberinto unicursal o labyrinth: es aquel en el cual solo existe un único camino desde la entrada hasta la parte central, sin posibles aberturas ni elecciones.

Laberinto multicursal o maze: es aquel en el cual existen varios caminos desde la entrada, los cuales muchos de ellos terminan en callejones sin salida, otros en bucles, otros llevan directos a la salida o punto objetivo y otros son la dirección contraria.

Estos tipos de laberintos a su vez son divididos a su vez en diversas subcategorías.

2.3.1 Laberintos Unicursales

Tal y como se indica en [1] y en [2] existen varios subtipos de laberinto unicursal dependiendo de la forma en la cual está construido el laberinto haciendo un resumen de dichas referencias se puede obtener que, entre los más destacados o conocidos están:

- **Clásico o Cretense**: "laberinto con forma ovoide" de 7 circuitos concéntricos. Es de diseño sencillo.

- **Clásico Báltico**: posee un centro y dos entradas las cuales se juntan para llegar al centro.

- **Romano**: originalmente era de forma cuadrada dividido en cuadrantes. Con el tiempo fue adquiriendo también forma de círculos concéntricos, también divididos en cuadrantes. Dependiendo del punto en donde empiece el laberinto este puede ser espiral, meandro o serpentino.

- **Medieval**: No posee ningún estándar de forma y se solían usar para decorar iglesias. El número de elementos concéntricos suele variar entre 6 y 11 y empiezan a utilizarse deformaciones en las esquinas del laberinto generando figuras.

- **Actuales**: Estos laberintos combinan la estética de los anteriormente mencionados, generando obras de arte modernas.

2.3.2 Laberintos Multicursales.

Con el paso del tiempo, desde la época greco-romana, la forma de los laberintos fue cambiando, en especial la de los del tipo multicursal haciéndolos más enrevesados, siendo empleado sobretodo en la jardinería a partir de la edad media.

Tal y como se indica en [1] y en [3] existen varios subtipos de laberinto multicursal dependiendo de la forma en la cual está construido el laberinto. Haciendo un resumen de las referencias indicadas con anterioridad, los más destacados o conocidos son:

- **Laberintos de conexión simple**: Son los más conocidos pues aun dependiendo de la complejidad que pudiese tener su diseño este tipo de laberintos se caracterizan por ser capaz de encontrar la salida o el punto que se está buscando si se recorre manteniendo la mano siempre a una de las paredes, dando igual si es la izquierda o la derecha pues siempre se haya la solución, la Figura 1 es un ejemplo de este tipo de laberinto.

- **Laberintos de conexión múltiple**: debido a que la técnica de la mano en la pared hacia que la dificultad del laberinto no fuese mayor que la de uno unicursal, por lo que a principios del siglo XIX se desarrollaron laberintos los cuales no poseían barreras interconectadas entre sí, lo que daba forma de islas en donde el objetivo se encontraba dentro, haciendo más complicado hallar al mismo. Un ejemplo sería el de la Figura 3 .

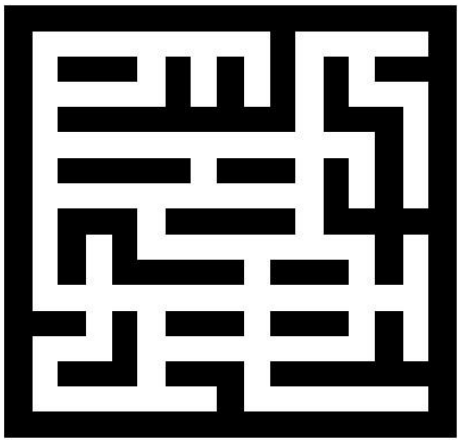


Figura 3: Ejemplo de un laberinto de conexión múltiple.

- **Laberintos tridimensionales**: esta técnica se empezó a utilizar en el siglo XIX en dibujos, sin embargo, hacia finales del siglo XX es cuando se empiezan a desarrollar para jardines, empleando elementos como son puentes como único vinculo entre islas y pasos inferiores para así generar mayor dificultad.

- **Laberintos interactivos**: empleados en sitios turísticos y centros de ocio. Estos laberintos van cambiando ciertas barreras mediante dispositivos electrónicos a medida que los visitantes los van recorriendo, por lo que hacen que según avances se cierre el camino por el que has estado, o si ves la salida al llegar a cierta zona mediante sensores te bloqueen en camino y tengas que buscar otra salida, etc.

- **Laberintos de movimiento condicionado**: son populares dentro del ámbito educativo, pues las diversas elecciones posibles no vienen dadas por la forma del laberinto, sino por las instrucciones que están contenidas en su interior, en donde te dicen si puedes avanzar y que posibles caminos tienes, algo así al juego de la oca pero con un laberinto multicursal en vez de ser uno unicursal.

2.4 Historia

Los primeros indicios de laberintos aparecieron en la prehistoria, más concretamente en la Edad de Bronce, en donde se han visto reflejados en los restos arqueológicos de aquella época que se han obtenido, como son en algunos tallados en roca.

Más adelante, según textos antiguos en Egipto existió un Gran Laberinto, el cual en el siglo II d.C. se perdió su ubicación, sin embargo, en el 2008 gracias a las expediciones arqueológicas y a los escáneres con las nuevas tecnologías, se ha conseguido volver a localizar este monumento.

También han llegado hasta nosotros textos y documentos de gente que lo presencio y que nos lo describen como era en aquella época, por lo que sabemos a ciencia cierta que éste es el laberinto al que se referían las escrituras y no otro distinto.

Se decía que estaba construido en piedra y adornado con relieves e imágenes y que propósito principal era para ofrecer al rey protección contra sus enemigos como de la misma muerte y según Heródoto poseía 1500 cámaras superficiales y muchas más subterráneas.

En la mitología griega esta el laberinto de Greta el cual según la mitología fue construido para albergar al minotauro, un ser mitad hombre y mitad toro. En este laberinto se desechaba a la gente a su suerte para que fuesen victimas del minotauro.

En el imperio romano los laberintos fueron comúnmente usados, sobre todo a modo de mosaicos para el suelo, originando diversos estilos y variedades de laberinto, pasando del cuadrado al circular y a usar diversos tipos de inicio.

A partir de los siglos IX y X el cristianismo empieza a emplear laberintos de cuatro lados simétricos con el objetivo de decoración de manuscritos, paredes, suelos de iglesias y algunas edificaciones, como es la Catedral de Chartres, llamada también Laberinto Chartres.

A medida que pasan los siglos los laberintos se deforman por las esquinas, dando lugar a nuevos tipos de figuras jamás vistas hasta entonces, las cuales empiezan a usarse para jardines, por ejemplo, en Paris.

Finalmente en la actualidad con el nuevo resurgimiento de los laberintos como un concepto de arte en sí mismo, se han desarrollado nuevos estilos y técnicas de laberintos, que van desde el minimalismo, los cuales contienen los suficientes recovecos para dar la impresión de laberinto pero sin ser complicados, hasta diseños complicados en los cuales se superponen arte y conceptos matemáticos, arquitectónicos, etc.

2.5 Aplicaciones de los laberintos en la actualidad.

Los laberintos no solo se emplean en la actualidad como en elemento decorativo para jardines o mosaicos, con el tiempo se ha ido observando cuan útil puede resultar este tipo de figura. Algunas de sus aplicaciones hoy en día son:

- **Elemento decorativo**: aunque se ha comentado que no solo se emplean como un elemento decorativo, cabe destacar que las construcciones que se realizan son impresionantes, ya sean laberintos tridimensionales o más realizados a partir de complicados cálculos matemáticos, haciendo unas bellas obras de arte.

- **Elemento de ocio**: se utiliza sobre todo en parques de diversiones, como en las salas de los espejos, mansiones del terror y otros recorridos interactivos que cambian a medida que los visitantes se van adentrando en él para asegurar entretenimiento y desorientación, mientras que a veces con un guía los van llevando por el recorrido.

- **Aplicaciones médicas**: los laberintos se utilizan desde edades tempranas para comprobar si un niño o persona tiene déficit de atención o algún trastorno neuronal y gracias a ellos permiten evaluar el nivel del daño cerebral en la persona afectada.

- **Robótica:** mediante el diseño de algoritmos, un robot puede detectar con sus sensores lo que hay a su alrededor y hacerse su propio laberinto con una posición inicial en la cual es donde está y una posición final que es el destino al cual pretende ir, gracias al algoritmo y a los sensores que le indican lo que hay a su alrededor se genera un laberinto el cual es capaz de recorrer y así poder ir a realizar sus labores de por ejemplo salvamento, mantenimiento, etc.

- **Industria de videojuegos**: En la industria de los videojuegos se llevan utilizando algoritmos diversos para generar laberintos casi desde los inicios de los mismos, ya sean para juegos de tipo aventura gráfica, como los que son juegos de tipo RPG en los cuales se usaba la consola de comandos como interfaz grafica, como se puede ver en la Figura 4.

  Si bien es cierto, muchas veces estos juegos en la actualidad simplemente son creados a partir de un editor gráfico, como es el RPG-Maker u otros motores, los cuales ya diseñan un laberinto estático, existen otros de tipo indie como es el estilo de Enter The Gungeon en el cual sus niveles son únicos en forma de laberinto, pese a que obligatoriamente necesitan que existan ciertas salas pero no indican que posición debe tener Se puede ver un ejemplo de cómo son estas salas en la Figura 5.

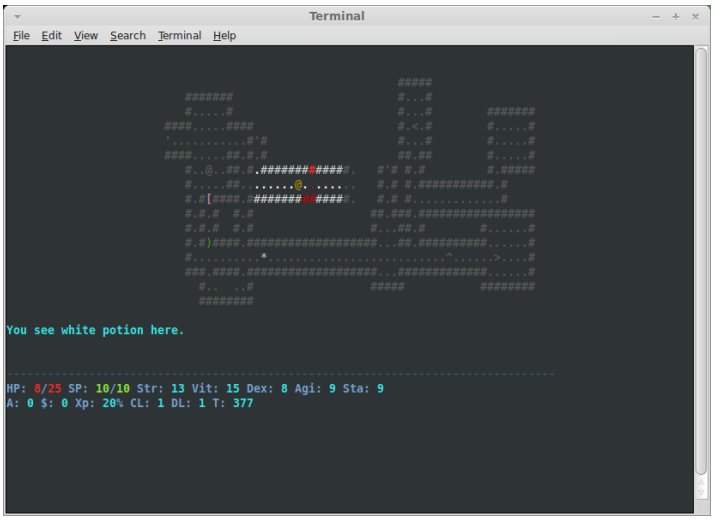


Figura 4: Laberinto por terminal.

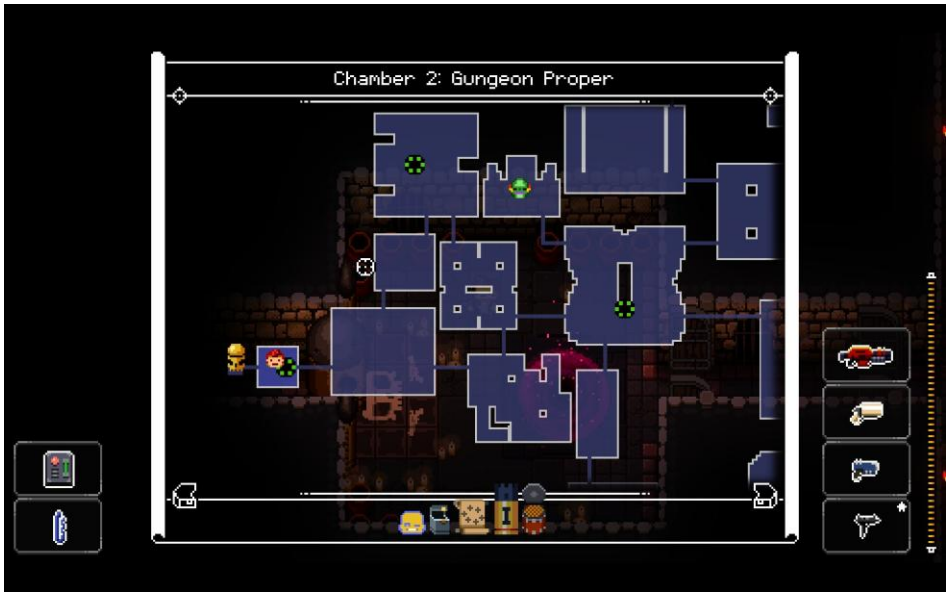


Figura 5: Ejemplo de generación de una sala de Enter the Gungeon

3 Estado del arte

A lo largo del anterior apartado se explicó en qué consistían los laberintos y que tipos de laberintos nos podíamos encontrar con mayor probabilidad, pues solamente se referenciaron los más comunes, y se realizo un rápido recorrido por la historia de estos, junto con las aplicaciones y usos más comunes que poseen en la actualidad.

En este apartado se va a centrar en el ámbito que nos ocupa: la informática, aparte, se hará una descripción acerca de los algoritmos que se utilizan en nuestro ámbito y cuáles son los más comunes.

Finalmente, se realizarán análisis de distintos lenguajes y sus códigos para generar laberintos que se pueden encontrar.

3.1 Laberintos y la tecnología

La generación de laberintos antes era sumamente costosa de realizar, pues aunque fuese solo para hacerlo en papel, se tenía que realizar mentalmente y cuanto mayor era el laberinto mayor tiempo se demoraba, llegándose a tardar desde horas o días hasta semanas o meses, pues dependiendo del tipo de laberinto (si es el más común) puede que requiera que toda la zona este rellena de tramos los cuales se vuelven encrucijadas o no.

Gracias a la tecnología este tipo de dificultades han podido ser resultas en mayor o menor medida, y si bien es cierto que no existe una única solución que satisfaga las necesidades de todo el mundo, si que existen diversas soluciones las cuales poseen características que se adecuan mas a ciertas necesidades y es el diseñador o usuario el que debe saber en qué momento debe emplear cada una de estas técnicas.

3.1.1 Clasificación los tipos algoritmos según su generación.

Los algoritmos se pueden clasificar en distintos tipos y subtipos, dependiendo de las características que poseen a la hora de generar un laberinto, pues tal y como existen diversos tipos de laberinto y solo se ha hablado de los mas genéricos y conocidos en el presente documento, se realizará una indicación de tipos de algoritmos más conocidos, pues sino el documento solo consistiría en hablar de las diferentes ramificaciones.

Según la referencia [4] se puede hacer una primera clasificación de los laberintos en 7 grupos distintos cada uno con varios subgrupos distintos, los cuales serían haciendo un resumen de la referencia [4]:

- **Dimensión:** indica en cuantas dimensiones está basado el laberinto. Los algoritmos de esta clasificación son los que más se suelen utilizar y que más existen para la

generación de laberintos, por lo que se ahondara en más detalle y más tarde se hablarán de algunos de ellos.

- **2D**: Muchos de los laberintos existentes, ya sean generados por computador, creados en papel o en elementos decorativos, como jardines, son en 2D, pese al aspecto tridimensional que pueda aparentar el uso de un objeto tridimensional como es un muro, desde una perspectiva aérea se sigue considerando de 2D.

  Los algoritmos empleados para el Asset de Unity son de este tipo, ya que aunque utiliza elementos en 3D, desde una perspectiva aérea se pueden considerar de 2 dimensiones pues no poseen plantas. Más adelante se mostrarán algunos ejemplos de pseudocódigo de este tipo de generación.

- **3D**: No implican que se empleen necesariamente elementos en 3D, sino que se caracterizan por ser laberintos con varios pisos en donde se puede moverse no solo por las coordenadas norte, sur, este y oeste sino también por la altitud, pudiendo subir y bajar de piso a voluntad siempre que el laberinto lo permita.

  La implementación más común de este tipo de laberintos es a partir de una lista de los laberintos en 2D junto con elementos que permitan la subida y bajada entre pisos, como son por ejemplo unas escaleras.

- **Higher dimensions**: es una extensión del algoritmo para generar laberintos en 3D, en el cual aparte de tener las coordenadas y la altitud, también se caracteriza por incluir una serie de condiciones o portales los cuales te permiten cambiar entre planos, tiempo, multiverso o lo que el desarrollador del laberinto desee incluir.

- **Weave:** tipo de laberinto en donde se solapan laberintos en 2D mediante puentes, nexos o cualquier elemento que te permita unirlos, por lo que al final aunque son varios laberintos distintos, están todos unidos en un mismo piso y mapa y se puede alcanzar desde cualquier punto otro punto distinto, aun siendo de otro laberinto.

- **Hiperdimensión**: Hace referencia a las dimensiones de todos los elementos y objetos dentro del laberinto los cuales se mueven a través del mismo, a diferencia del propio laberinto.

  - **No-Hypermaze**: la mayor parte de los laberintos son de este tipo, ya que solo hay una cantidad fija de movimientos posibles para un laberinto concreto.

  - **Hypermaze**: simula un laberinto interactivo, por el que a medida que se mueve el laberinto se va trasformando, haciendo que siempre haya infinitos movimientos posibles.

- **Topología**: indica la geometría del espacio del laberinto. Es normal si está en un espacio euclidiano o es planar si posee una topología anormal fuera del espacio euclidiano, como es por ejemplo la cinta de Moebius.

- **Tessellation**: es la división del espacio que contiene el laberinto en polígonos. Esto da pie a que un laberinto no sea rectangular, sino que tenga diversas formas: circular, triangular, hexagonal, etc., dependiendo del número de conexiones que posea cada celda.

- **Enrutamiento**: Indica qué tipo de calles genera el algoritmo del laberinto. Este tipo de clasificaciones es importante porque dependiendo de cual se elija, el laberinto podrá ser perfectamente accesible desde todos los puntos, podrán existir zonas no accesibles etc. Los más representativos son:

    - **Perfecto**: Una descripción según [6] es: "Se define un laberinto perfecto como un laberinto el cual tiene un único camino desde cualquier punto del laberinto a cualquier otro. Esto implica que el laberinto no posee secciones inaccesibles caminos circulares o áreas cerradas". Este es uno de los tipos de algoritmo que se han diseñado para el Asset de Unity

    - **Braid**: Laberinto multiconectado el cual no posee ningún callejón sin salida. En la Figura 6 se puede ver un ejemplo de este tipo de laberinto.

    - **Unicursal**: Como ya se vio anteriormente posee un único camino, sin callejones.

    - **Sparseness**: Laberinto que posee secciones inaccesibles o áreas cerradas. Uno de los algoritmos diseñados realiza la generación de este tipo de laberintos, más adelante se explicará cómo.

    - **Braid parcial**: Realiza una mezcla entre un laberinto multiconectado con uno perfecto, dando lugar a un laberinto que posee bucles en sus caminos a la par de generar caminos sin salida. Este tipo de algoritmo también es uno de los implementados para el Asset de Unity.

- **Textura**: indica a la hora de generar el laberinto el tipo de forma que acaban tomando los caminos, ya que aunque cada algoritmo genera caminos aleatorios la aleatoriedad al final como está centrada a seguir ciertas reglas acaba formando siempre unas figuras los caminos mu características del algoritmo, como es uno de los algoritmos implementados en el Asset que los caminos tienen formas escalonadas como se puede apreciar en la Figura 7.

- **Enfoque**: Define cómo empieza el laberinto, si suponemos que es un laberinto vacío y se van insertando los muros o si a partir de una malla con todos los muros insertados, estos se van eliminando poco a poco gracias al algoritmo. El ultimo método de ir eliminando muros es el que se emplea dentro del Asset de Unity para los distintos algoritmos de generación, sin embargo, debido a las características de Unity se le hicieron ciertas modificaciones para sacarle un mejor rendimiento que se explicaran con más detalle en próximos apartados.

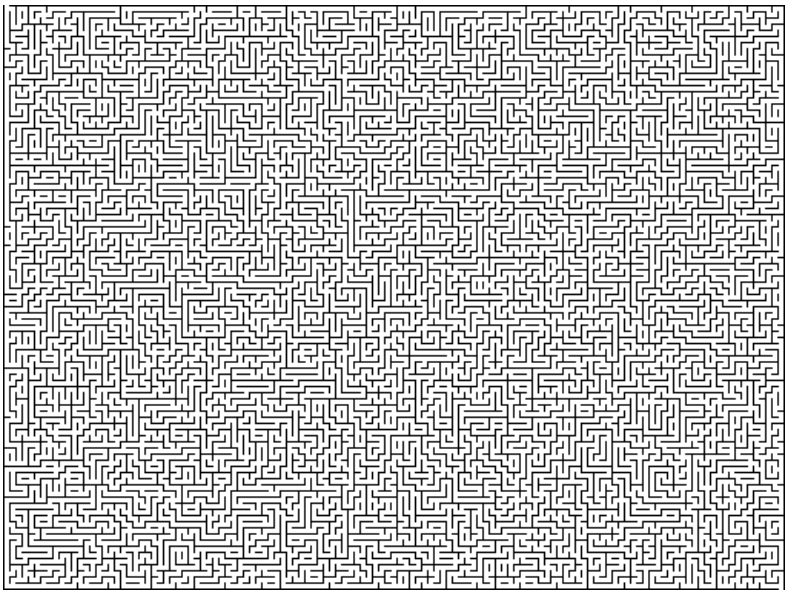


Figura 6: Laberinto Braid.

Figura 7: Laberinto generado con Asset de Unity

3.1.2 Algoritmos más comunes.

Los algoritmos para generar laberintos son muchos y solo se enunciarán los que mayor relevancia tienen por ser los más ampliamente extendidos o de mayor eficiencia, por lo que no se hablará de todos los algoritmos para todos los tipos de laberintos que existen. Estos algoritmos son la base de la generación de laberintos más complejos como son los de tipo multiconectado o los de tipo 3D entre otros.

- **Algoritmos de generación perfecta**: como se dijo anteriormente no poseen ciclos en los caminos ni áreas cerradas y solo hay un único camino desde un punto hasta otro. Algoritmos que realizan este tipo de generación son los siguientes:

- **Algoritmo de Búsqueda en profundidad**: Base de uno de los algoritmos empleados en el Asset de Unity. Utiliza la teoría de grafos para generar el laberinto, por el cual a medida que los va visitando el árbol de entre todos los nodos selecciona el que mejor se adecua a su estrategia, que en el caso de los laberintos como se desea que sean arboles únicos es seleccionar uno al azar y mantiene en memoria el registro de los que ya ha visitado para volver atrás y va avanzando en la profundidad del árbol hasta que se encuentra en un punto en el que ya no tiene más nodos por expandir. Para hacer esto posible, se vale de una matriz dividida en x cantidad de celdas, cada una de estas celdas posee un número constante de muros, en el laberinto del cual se va a centrar en laberintos rectangulares debido a que el Asset es de tipo rectangular, por lo que son 4 los muros: norte, sur, este y oeste. También posee por cada celda un flag para indicar si ha sido visitada o no.

```
Obtener una celda aleatoria.
Celda actual = celda aleatoria.
Celda actual = visitada.
Repetir
      Si hay vecinos no visitados
            Obtener una celda vecina de celda
      actual no visitada.
            Eliminar muro
            Celda vecina visitada = true
            Pila += celda actual
            Celda actual = celda vecina.
      Sino
            Extraer celda de la pila
            Celda actual = celda de pila

Mientras pila isEmpty
```

Debido al hecho de que requiere de una pila para recuperar los nodos ya visitados tiene problemas con los recursos de memoria, por lo que para laberintos de un tamaño considerable no es útil este algoritmo.

- **Algoritmo recursivo con backtraking:** Basado en el algoritmo de búsqueda en profundidad, sin embargo, en vez de emplear la pila para recuperar nodos hasta el inicial y si saber que ha finalizado, se lleva un registro del número de nodos que faltan por visitar y la pila es un elemento para poder recuperar nodos, por lo que tiene menos propensión a sufrir una saturación de recursos que el algoritmo de búsqueda en profundidad. Es el que se emplea en el Asset cómo búsqueda en profundidad ya que implementa una generación de búsqueda en profundidad con retroceso. El pseudocódigo obtenido en [7] es el siguiente:

```
Obtener una celda aleatoria y marcarla como
inicial.
Celda inicial visitada= true.
```

```
Mientras existan celdas no visitadas
     Si la celda actual tiene algún vecino no
visitado
          Seleccionar un vecino al azar
          Insertar la celda actual en la pila.
          Eliminar el muro
          La celda actual = vecino seleccionado
          Celda actual visitada = true
     Sino si la pila no es vacía
          Extraer celda de la pila
          Celda actual = celda de pila
```

- **Algoritmo de Kruskal**: Este tipo de algoritmo también se basa en mallas y celdas para su implementación. La idea es generar un árbol de expansión mínima, con lo que se van uniendo aleatoriamente pares de celdas adyacentes, de tal manera que el algoritmo va generando un laberinto que aparentemente parecen piezas sueltas, pero luego se van uniendo unas con otras hasta estar todas agrupadas en un único camino. También requiere un consumo de memoria para obtener todos los arboles de expansión mínima que se generan, por lo que el algoritmo si este tiene una malla muy grande puede ser problemático y costoso a la hora de unir los conjuntos de arboles de expansión mínima. Debido a estos inconvenientes este algoritmo no es de los implementados.

```
Hacer una lista con todos los muros
Insertar los muros en las celdas correspondientes.
Para todos los muros de i = rand
 Si el muro divide 2 celdas separadas
     Eliminar muro
     Unir al mismo grupo de celda las 2 celdas.
```

- **Algoritmo sencillo**: Existen algoritmos sencillos con poco coste de recursos y tiempo en comparación con otros, que si bien no se pueden insertar en una categoría concreta algunas de ellos, si que se pueden agrupar por tener la característica de ser eficientes.

  - **Árbol binario**: Este es otro de los algoritmos empleados para la generación de laberintos del Asset de Unity por su elevada versatilidad.

    Es uno de los pocos algoritmos que no requiere mantener el estado de los elementos del laberinto y ser capaz de generar un laberinto perfecto a costa de que genera siempre unas texturas muy características, como se puede observar en la Figura 7.

    Se basa también en una malla dividida en celdas de norte a sur y a lo largo del laberinto el algoritmo para obtener vecinos tiene que ser siempre el mismo y dar una diagonal norte-este, norte-oeste, sur-este o sur-oeste ya que sino el algoritmo que se genera es imperfecto y con áreas cerradas.

```
Para i=0; i< número de celdas; i++
        Seleccionar vecino al azar de Matriz[i]
        Eliminar muro
```

- **Árbol binario versión imperfecta:** Esta versión no genera solamente laberintos perfectos, sino que también genera de tipo Sparseness, en donde existen áreas inaccesibles.

  Esto se debe a que en lugar de tomar como posibles vecinos únicamente las mismas diagonales a lo largo de todo el laberinto, se pueden tomar los cuatro ejes cardinales, por lo que es posible abrir caminos que den una vuelta y no conduzcan a ningún lado y cuanto mayor sea la malla más probabilidades se dan.

  El algoritmo es el mismo que para el árbol binario pues solo cambian los vecinos que obtiene.

4 Análisis y Diseño

A lo largo de este apartado se describirán el análisis con los requisitos requeridos del Asset y requisitos requeridos de la implementación y el diseño seguido para el Asset y de la implementación.

4.1 Análisis

Se especificará qué comportamiento o requisitos debe poseer las funciones del Asset, concretizando una única característica por funcionalidad lo mas concisa y claramente posible. Los tipos de requisitos son dos: funcionales y no funcionales.

4.1.1 Requisitos funcionales.

Son el conjunto de comportamientos, entradas y salidas que se espera obtener al emplear el Asset. Estos son:

Inicialización

Son los requisitos funcionales para las características de un laberinto.

Código	Requisito
INI-01	Ancho del laberinto variable a gusto del usuario, con valor por defecto a 5
INI-02	Largo del laberinto variable a gusto del usuario, con valor por defecto a 5
INI-03	Tamaño del muro para el ancho, alto y largo variables a gustos del usuario con valores por defecto.
INI-04	Vector de inicio del laberinto, en donde se inserta el primer muro inferior más a la izquierda, variable por el usuario. Por defecto en (0, 0, 0)
INI-05	Laberinto formado a partir de un número de Celdas fijado por el ancho y el largo indicados por el usuario, en el que solo están los muros insertados.
INI-06	JugadorAleatorio indica si el jugador se insertará en una celda aleatoria o en la salaInicio, es configurable por el usuario pero por defecto está a true.
INI-07	SalaInicio indica en que celda el jugador empieza si no es de tipo jugadorAleatorio. Modificable por el usuario, por defecto 0.
INI-08	GameObject que realiza la función de jugador, insertado por el usuario. Indica el punto de inicio dentro del laberinto.
INI-09	GameObject que realiza la función de muro, insertado por el usuario. Este se acoplará a las características dadas por el tamaño del muro.
INI-10	Terrain, insertado por el usuario para hacer de suelo, este objeto ya debe estar colocado dentro de la escena.
INI-11	Inicializador del gestor de laberintos, el cual genera una instancia del objeto. Utiliza como parámetros tanto el ancho, largo, muros, suelo, player, jugadorAleatorio y salaInicio como ningún parámetro.

Tabla 1: Requisitos funcionales de la inicialización del laberinto.

Generación:

Requisitos funcionales referentes a la generación del laberinto en sí.

Código	Requisito
GEN-01	Genera un laberinto según el tipo que el usuario desee: Búsqueda en profundidad, búsqueda en profundidad con multiconexión o árbol binario. La forma y características son las indicadas anteriormente por el usuario.
GEN-02	Borrar laberinto, deja la escena en un estado inicial.
GEN-03	Colocar terreno con el tamaño y posición del laberinto.
GEN-04	Creación de la malla con los muros de las celdas inicializados
GEN-05	Algoritmo búsqueda en profundidad
GEN-06	Algoritmo búsqueda en profundidad con multiconexión
GEN-07	Algoritmo de árbol binario
GEN-08	Obtener vecino no visitado
GEN-09	Obtener vecino visitado
GEN-10	Eliminar muro
GEN-11	Instanciar laberinto
GEN-12	El laberinto debe conservar los muros exteriores sin eliminarse durante la generación. No se generarán salidas ni entradas del laberinto.

Tabla 2: Requisitos funcionales de la generación del laberinto.

Elementos

Requisitos funcionales referentes la incorporación de elementos dentro del laberinto y modificaciones de las características de la navegación.

Código	Requisito
ELE-01	El usuario puede instanciar el objeto jugador dentro del laberinto y dependiendo de si tiene la característica de jugadorAleatorio o no se insertara en la celda inicial o en una aleatoria.
ELE-02	El usuario puede instanciar un objeto indicado dentro del laberinto en una sala especificada
ELE-03	El usuario puede instanciar un objeto indicado dentro del laberinto en una sala aleatoria
ELE-04	El usuario puede mover un objeto indicado en el laberinto a una sala especificada
ELE-05	El usuario puede mover un objeto indicado en el laberinto a una sala aleatoria
ELE-06	El usuario puede obtener unas coordenadas aleatorias pertenecientes a un punto accesible dentro del laberinto.
ELE-07	El usuario puede hacer que los muros que posean el valor NavMeshObstacle tengan la capacidad conseguir que las IAs autogeneradas de Unity las consideren muros infranqueables. Para ellos se debe hacer un Bake en el navigation del terreno.

Tabla 3: Requisitos funcionales de los elementos del laberinto.

4.1.2 Requisitos no funcionales.

Asset

Requisitos no funcionales del Asset en general.

Código	Requisito
ASS-01	Usabilidad: Asset sencillo, con funciones estándar que facilitarán al usuario que intente desarrollar con él el aprendizaje y manejo del Asset.
ASS-02	Usabilidad: pequeño manual de usuario añadido en la presente memoria.
ASS-03	Portabilidad: Debido a la característica de Asset, éste es portable se puede insertar como librería fácilmente dentro de cualquier proyecto de Unity, sea cual sea a partir de la versión 5.4
ASS-04	Escalabilidad: A partir de las funciones contenidas dentro del Asset se pueden generar nuevas funcionalidades para la creación de nuevas formas de laberintos y empleos de los mismos.
ASS-05	Rendimiento: Optimización de los recursos de RAM y GPU al máximo, para evitar inserciones de elementos del laberinto innecesarias y posibilidad de elegir entre 2 tipos de generación que cada uno necesita un rendimiento distinto al otro: - Búsqueda en profundidad con multiconexión: alto rendimiento - Búsqueda en profundidad: rendimiento medio - Árbol binario: muy bajo rendimiento

Tabla 4: Requisitos no funcionales del Asset generador de laberintos.

4.2 Diseño

Se hará una descripción de cómo está diseñado el Asset y en cómo se divide. Porqué se han seleccionado ciertos métodos sobre otros y se harán unos diagramas de clases mostrando los componentes del Asset y del Pacman.

4.2.1 Patrones de diseño

Los patrones de diseño empleados en el TFG son los siguientes:

- **Composite**: Como indican en [8] composición de un tipo de objeto simple para hacer otro con un grado de mayor complejidad. Empleado para añadir elementos de tipo Celda al objeto GestorDeLaberintos.

  Con ellos se crean el array de celdas y la lista de stackCelda.

- **Prototype**: Es el patrón que emplea Unity cuando mediante script se instancia un objeto en una posición, pues Unity ya tiene el objeto que se desea instanciar como componente y genera un clon de dicho componente en la posición indicada.

  La instanciación es el único método para insertar objetos, por lo que todos los muros, el jugador y objetos que se insertan gracias a las funciones implementadas, no son sino clones de un objeto al cual se hace referencia.

4.2.2 Diagrama de clases

La Figura 8 es el diagrama de clases del Asset, en el cual se pueden apreciar los scripts creados GestorDeLaberintos y Celda y las relaciones que tienen cada uno, así como las funciones que poseen.

También se puede apreciar la necesidad de ciertos elementos externos, que son propios de la interfaz grafica de Unity y que sirven para poder generar los gráficos. Estos son Terrain que es el suelo o terreno del laberinto y GameObject que son los muros y el jugador a insertar.

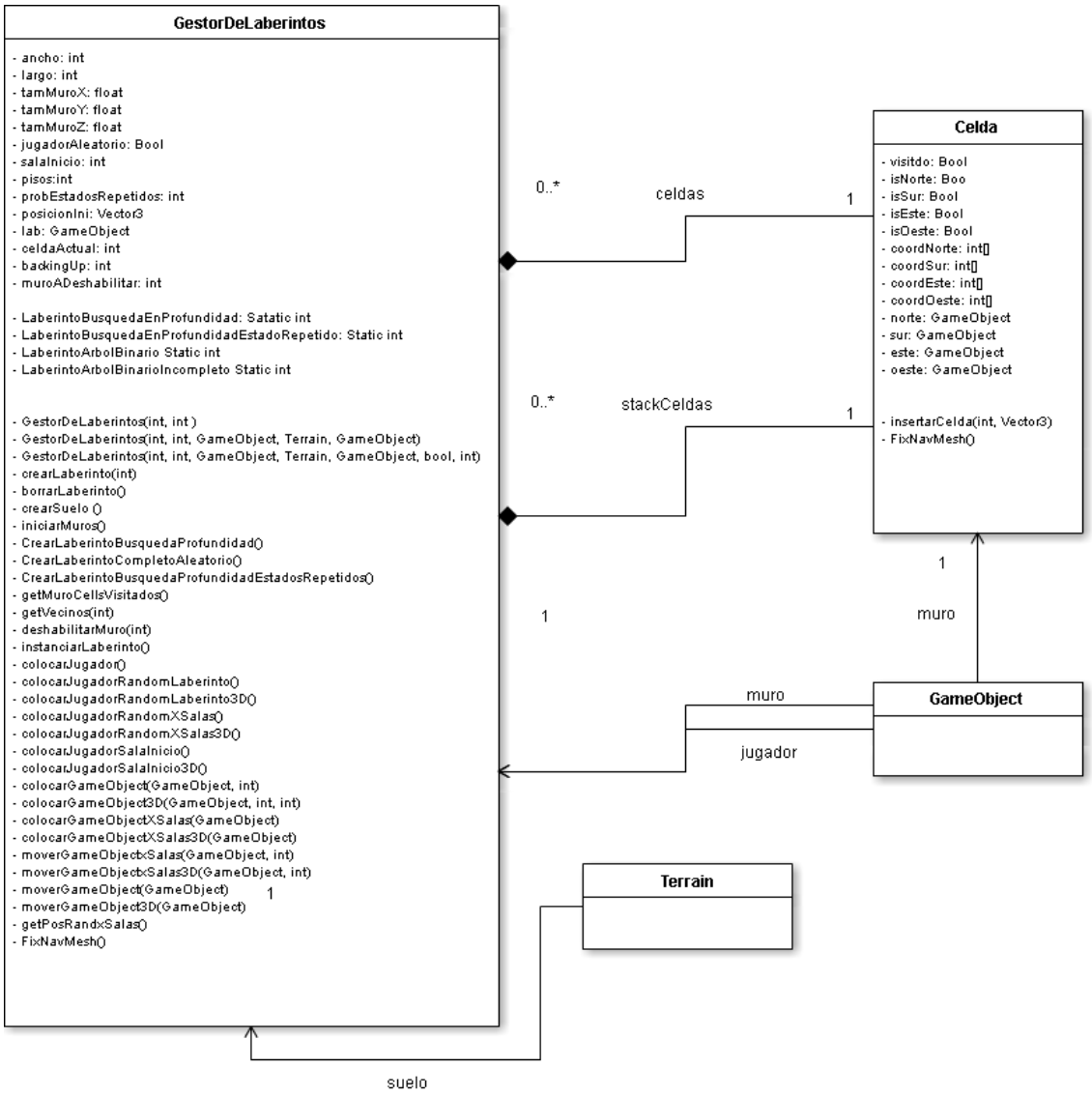


Figura 8: Diagrama de clases del Asset.

Como se puede apreciar en el diagrama de clases, existen dos objetos principales que son Celda y GestorDeLaberintos, estos son los scripts generados para la generación de laberintos.

- **Celda:** Gestiona las características que posee una celda, como es si esta ha sido visitada o no, los objetos de tipo muro que posee, las coordenadas de cada muro y sí el muro es insertable o no.

  Las funciones con las que cuenta son las de insertar los muros y la de gestionar la navegación por el mapa para las IAs en caso de que el usuario decida implementar alguna.

- **GestorDeLaberintos:** Gestiona las características que posee un laberinto, como es el tamaño, el jugador, el tamaño de los muros del laberinto, coordenada de inicio de laberinto, pisos que posee el laberinto, donde insertar al jugador, que probabilidades hay de bucles en los caminos y los tipos de laberintos existentes.

Las funciones con las que cuenta son en general las destinadas a crear un laberinto, insertar jugador, insertar objetos y gestionar la navegación de los muros del laberinto para las IAs en caso de que se deseen implementar alguna dentro del laberinto, tanto para laberintos con un único piso como laberintos con varios.

5 Desarrollo

En este apartado es explicará cómo está desarrollado el Asset y porqué se han tomado ciertas decisiones mientras se desarrollaba el mismo.

- El Asset está dividido en 2 scripts claramente diferenciados, como se vio en el diagrama de clases del anterior apartado, en donde se localiza en uno de ellos todo lo referente al contenido y manejar de una celda (script Celda.cs) y en el otro está todo lo referente a la creación y gestión de laberintos (script GeneradorDeLaberinto).

  Se ha decidido hacer el desglose en 2 tipos de objetos distintos y en diferentes scripts en primer lugar para mantener un orden y modulación en el Asset y en segundo lugar, para poder permitir al usuario instalar cualquiera de los 2 scripts como ya se verá más adelante.

- Para poder realizar el laberinto se ha hablado anteriormente que se puede crear o bien desde una malla e ir eliminando los muro o bien empezando sin nada, es decir estando todo vacío e ir insertando los muros.

  El primero es más fácil de programar, pero para objetos en 3D es ineficiente, en cambio la segunda forma es más complicada pero tiene la ventaja de resultar con mayor rendimiento.

  Para poder superar esta barrera, se ha desarrollado un algoritmo que genera todas las celdas de la malla, lo cual es algo común en ambos tipos de generación.

  A continuación, cada celda posee no solamente los cuatro GameObject muro como variables, sino que posee en Bool por cada muro. Estos al inicio de la creación del laberinto se inicializan a true, e indican si están que el muro de su coordenada debe ser insertado, en caso contrario no.

  Por otro lado, se indican las coordenadas del muro de cada celda con respecto a la coordenada de origen, la cual está abajo a la izquierda.

- Cuando los algoritmos de generación de laberintos deciden que un muro es prescindible y deciden desecharlo, simplemente se comprueba que no hay problema con eliminar el muro e indican en la celda el valor del Bool de ese muro a false.

  Finalmente, cuando el algoritmo ya ha terminado de procesar toda la malla, se instancia el laberinto, insertando en la escena los muros que se indicaron como elementos true, mientras que aquellos que se indicaron como false no se instancian.

De esta forma, se obtiene un rendimiento en gráfica y RAM similar al método de ir insertando muros, pero con la facilidad del algoritmo de ir eliminando muros.

- Otra de las características es la obtención de vecinos, la cual está condicionada por el hecho de que aparte de que el siguiente vecino no debe haber sido visitado, tampoco puede ser el muro uno de los exteriores del laberinto. Otro de los elementos es la pila en la cual se van insertando los elementos según se van visitando en el algoritmo y el vecino los va extrayendo para cuando no existe ningún vecino disponible.

- Para generar los laberintos de tipo braid o braid parcial se ha empleado el algoritmo de DFS y en la comparación que realiza de si el vecinoActual esta visitado mediante un sistema de probabilidades que van del 0 al 99 y luego se suma el valor insertado por el usuario de probEstadosRep, si la suma total es igual o superior a 100 entonces llama a otra función para buscar vecino pero esta vez que si haya sido visitado.

  De esta manera se obtiene un vecino que ya había sido visitado pero desde una posición distinta, la única diferencia es que en este caso la celdaActual no será el vecino obtenido, sino que se seguirá con el algoritmo del DFS como si no hubiese pasado nada.

- Para el diseño de los laberintos en 3D con distintos pisos, se ha optado por la alternativa de no incluir suelos ya que estos opacarían el mayor porcentaje del mapa y para mostrar una generación en 3D no es útil y práctico.

- Para evitar problemas con la gestión de la inserción de elementos, ya que el GestorDeLaberintos posee tantas variables, se ha optado por no realizar un control de errores salvo en un par de casos que es para seleccionar el tipo de laberinto y para crear el laberinto el mínimo de pisos si es menor que 1 este se toma como 1.

  Para los demás casos se ha evitado el uso de generar el control, pues se retornaría un valor nulo y el usuario no sabría el porqué de todos sus elementos y en qué punto se produce. Aparte, gestionar los objetos de Unity es muy difícil pues tienen un montón de parámetros los cuales son dinámicos y tienen su propia gestión.

6 Integración, pruebas y resultados

El este apartado se comprobará el correcto funcionamiento del Asset, para ello es necesaria su implementación en aplicaciones para su correcta visualización, gracias a ello se podrán realizar las pruebas para determinar sí esta correcto o no.

6.1 Implementación

Se han realizado varias implementaciones del Asset en diversas aplicaciones para comprobar el correcto funcionamiento del mismo, estos son:

Generador:
Con esta implementación, se pretende comprobar que se generan y borran correctamente los laberintos.

Al pulsar el botón crea un laberinto aleatorio, de tamaño aleatorio entre 3 y 20 y del tipo aleatorio.

Patrulleros:

En esta implementación se dispone de un número aleatorio de enemigos que se van moviendo de manera libre por el laberinto de cantidades entre 1 y 10, posee un objeto llave y un objeto salida las cuales se colocan en el mapa de manera aleatoria igual que el jugador y los enemigos, una vez que se tenga la llave se gana el juego.

Con esta implementación ya no se pretende testear la generación del laberinto, el cual ya ha sido comprobado, lo que se desea es comprobar cuan efectiva es la inserción de elementos dentro de un laberinto, del jugador y el correcto funcionamiento de la función navigation que ofrece Unity para la creación de IAs.

Pacman:

Con esta implementación se pretende testear que se puede recrear un juego “sencillo” y no una funcionalidad de algún juego o mini juego. Se ha añadido más dificultad al número de inserciones de objetos y de sus interacciones que en las anteriores implementaciones para comprobar que es posible la creación de una aplicación con un nivel de programación más elevado.

6.2 Pruebas y resultados.

Las pruebas que se han realizado dependen de la implementación del Asset, por lo que los resultados varían también. Estos son los siguientes:

Laberintos:

Las pruebas que se realizaron fueron:

- El terreno no esté descolocado con respecto al laberinto. Figura 27
- El terreno esté acoplado en tamaño al laberinto. Figura 20, Figura 21, Figura 22, Figura 23 y Figura 24
- No existan huecos entre los muros. Figura 25 y Figura 28
- La inserción de muros sea la adecuada. Figura 28
- El tipo de generación se adecue al tipo de algoritmo, que aunque es aleatorio, se sabe los tipos que hay y por ejemplo, lo que se busca es que no existan errores del tipo áreas cerradas en laberintos que no son Árbol binario versión imperfecta o bucles en aquellos que no son de tipo DPS braid. Figura 20, Figura 21, Figura 22 y Figura 24
- Generación de pisos. Figura 23
- Lo muros exteriores no sean eliminados por culpa del algoritmo. Figura 26
- El tamaño esté entre el rango de valores dado. Figura 29

Los resultados satisfacen todas las pruebas como se puede apreciar en las imágenes ubicadas en el Anexo D: Imágenes de los test.

Patrulleros:

Las pruebas realizadas están dirigidas a comprobar cuan buenos son los algoritmos de inserción de objetos aleatorios, jugador y navegación:

- Se inserta correctamente el jugador en una posición aleatoria del laberinto, en el centro de una celda y cada vez que se pierde una vida se vuelve a la posición original. Inicio en Figura 35, después de perder vida en Figura 36
- Se insertan GameObject dentro del laberinto en una celda aleatoria en la parte central. Figura 30, Figura 31 y Figura 35.
- IAs con movimientos aleatorios no tienen dificultades para ir recorriendo el laberinto. Figura 35 y Figura 36.
- Se puede desplazar el jugador por el laberinto. Figura 37.

- Se comprueba que la colocación del GameObject y del jugador esté justo encima del terreno y no debajo o entre medias del mismo. Figura 30, Figura 31, Figura 32 y Figura 33.

- Se comprueba que se obtienen posiciones aleatorias correctas dentro del mapa, las cuales son las posiciones a las que se mueven los enemigos. Figura 35, Figura 36 y Figura 37.

- Se comprueba que el navigation que sea correcto pausando en juego en el editor Unity. Figura 34.

No se incluyen pruebas del tipo que no se caiga del terreno por que este no tiene colisión, pues no pertenece al laberinto sino al que inserta el terreno como elemento. Lo mismo sucede con muros, si estos se pueden atravesar no se realizan pruebas pues pertenece a la implementación y no al Asset. Tampoco se incluyen pruebas realizadas en laberintos.

Los resultados satisfacen todas las pruebas como se puede apreciar en las imágenes ubicadas en el Anexo D: Imágenes de los test.

Pacman:

Aparte de las pruebas realizadas para las patrullas las pruebas realizadas están dirigidas a comprobar los algoritmos de inserción de objetos en una celda concreta y la navegación de IAs más complejas.

- Comprobar que se insertan correctamente los Enemigos de tipo GameObject en las celdas especificadas. Figura 40

- Comprobar que se insertan correctamente las xp y las esferas de invencibilidad en las celdas indicadas. Figura 39

- Las IAs de los fantasmas se pueden mover por el laberinto tal y como se movían las personalidades del juego original. Figura 41 y Figura 42

No se incluyen pruebas del tipo que no se caiga del terreno por que este no tiene colisión, pues no pertenece al laberinto sino al que inserta el terreno como elemento. Lo mismo sucede con muros, si estos se pueden atravesar no se realizan pruebas pues pertenece a la implementación y no al Asset.

Los resultados son satisfactorios salvo para las IAs, que con la personalidad tan compleja en 3D tienen alguna que otra dificultad, pero esto no es del navigation que se haya generado mal sino que son IAs sencillas que se tienden a colapsar. Se pueden ver los resultados en las imágenes ubicadas en el Anexo D: Imágenes de los test.

7 Conclusiones y trabajo futuro

7.1 Conclusiones

Durante la carrera en la asignatura de videojuegos, aprendí a utilizar Unity y a usar el entorno gráfico para insertar elementos dentro de una escena y a esos elementos añadirles con el editor gráfico scripts para que realizasen ciertas acciones. Gracias a esto pude realizar el TFG pues ya existía una base de conocimientos con respecto a la plataforma. También me ha ayudado todas las asignaturas de programación, pues me dieron la soltura suficiente para desenvolverme y saber en qué sitios buscar información y ayuda.

A lo largo de este trabajo se han ido encontrando diversas dificultades debido a que Unity no provee de demasiada ayuda con respecto a la creación de aplicaciones mediantes scripts únicamente y por desgracia es en lo que se basa esencialmente mi aplicación, si bien es cierto que utiliza objetos como muros, jugador y objetos a insertar en el laberinto estos no están insertados, los tiene que insertar el script, jugar con su posición, tamaño, etc., lo cual es algo que si bien existe una API con todas las características que posee un objeto, no es tan fácil de programar como en el C# aunque supuestamente se esté programando en ese lenguaje, un ejemplo se puede observar a la hora de crear un objeto:

En C#

GestorDeLaberintos gdl = new GestorDeLaberintos(ancho, largo);

Muy intuitivo, como se ha visto en los lenguajes orientados a objetos, pues haciendo esto en Unity en un script C# no te dice que esté erróneo te lo permite hacer pero el objeto es null. Conclusión, todo lo dado este año de C# se me ha ido un poco al traste pese a que supuestamente es C# y lo mismo le ocurre a los script js que dicen ser javascript pero que no.

Por si alguien quiere saber cómo crear un objeto en Unity sería del siguiente modo:

GestorDeLaberintos gdl = new GameObject.AddComponent<GestorDeLaberintos>(ancho, largo);

Hasta que finalmente encuentras que estás creando mal el objeto, encuentras en algún sitio cómo crear el objeto porque la gente no crea objetos simplemente indica que tipo quiere el objeto lo pone público y luego ya con la interfaz añade el objeto, por lo que no se preocupa por estas cosas, transcurre bastante tiempo.

Por motivos como el que se ha descrito es por el que si tuviese que hacer el TFG con los conocimientos que tenía en ese momento tal vez me lo replantease, pues la dificultad, pese a que ya tenía cierto nivel en Unity, ha sido algo elevada.

Aparte de esas dificultades que han ido surgiendo y que me han atascado más que nada, me han hecho darme cuenta y apreciar cuan necesario es un generador de laberintos, pues si yo que ya había trabajado con scripts y generado ciertas funcionalidades en la asignatura de

videojuegos, una persona que tiene un nivel de conocimientos bajo no tendría más alternativa que realizar los laberintos a mano, y bueno a mano está bien si es solo un puzle de un único uso pero si deseas innovar es preferible el generado.

Mi conclusión final es que ha sido bastante sufrido, pero me ha hecho darme cuenta debido a la dificultad que he tenido que si dejo esto como código libre personas que no tienen la capacidad suficiente para programar tan al detalle en Unity podrán crear grandes aplicaciones sin tanta dificultad.

7.2 Trabajo futuro

Como cualquier aplicación creada, siempre existen aspectos a mejorar o elementos que se podrían incluir y que por falta de tiempo, dificultad o por no haber caído en la cuenta de ellos antes no se realizo en su momento. Estas mejoras a realizar están presentes tanto en el Asset como en el Pacman.

Para estos casos existen las versiones de los programas, también conocidos como DLC en el mundo de los videojuegos, pues ya son versiones funcionales. Las mejoras que me gustaría realizar son:

Asset

- Incluir algoritmos de generación de laberintos de tipo unicursal.
- Incluir laberintos que no sean únicamente rectangulares.
- Incluir laberintos interactivos.
- Poder especificar el número de entradas y salidas que posee el laberinto y que no sea uno cerrado, ya que tal y como está actualmente si no se restringe el algoritmo genera la cantidad de entradas y salidas aleatorias que desee y así se evita.
- Añadir algún algoritmo generación más.
- Añadir suelos entre los pisos con espacios vacíos en algunas zonas de manera aleatoria, que no se añadió por temas de visibilidad y diseño.

Pacman

- Incluir elección de nivel de dificultad.
- Añadir niveles, que no sea solo un único nivel de juego y si te lo pasas ganas. A medida que se aumentan los niveles la probabilidad de que se generen bucles en los caminos disminuye, aumentando así la dificultad.
- Hacer las IAs en 3D en condiciones y no con las opciones que proporciona Unity pues esto hace que el movimiento cuando cambian los fantasmas entre huir y perseguir a Pacman sea algo errático. No se podían generar unas IAs que no fuesen

simplemente de movimiento aleatorio ya que como el laberinto cambia por juego y posee bucles no se pueden generar buenas condiciones con el 3D, requeriría un proyecto propio.

- Incorporar al cuarto fantasma que no se incluyo por no saber con exactitud como plasmar su movimiento del código original con respecto a la interfaz de IAs que provee Unity.

- Mejorar los gráficos.

Referencias y Bibliografía

[1] **"Tipos de Laberintos",W3C Visitado el 15 Mayo 2016** http://laberintos.weebly.com/tipos-de-laberintos.html

[2] **"Laberinto", W3C Visitado el 1 Junio 2016** https://es.wikipedia.org/wiki/Laberinto

[3] **"Laberintos Multicursales", W3C Visitado el 10 de Junio 2016** http://www.labolab.net/tipos/laberintos-multicursales/

[4] **"Think Labyrinth", W3C 23 Septiembre 1996, Visitado el 13 de Junio 2016**

[5] http://www.astrolog.org/labyrnth/algrithm.htm

[6] **"Step by step perfect maze generation with php", W3C 20 Agosto 2006 , Visitado el 18 Mayo 2016** http://www.emanueleferonato.com/2006/08/20/step-by-step-perfect-maze-generation-with-php/

[7] **"Maze generation algorithm", W3C Visitado el 20 de Febrero 2016** https://en.wikipedia.org/wiki/Maze_generation_algorithm

[8] **"Composite (patrón de diseño)", W3C Visitado el 24 de Junio de 2016** https://es.wikipedia.org/wiki/Composite_(patr%C3%B3n_de_dise%C3%B1o)

[9] **"Origen e historia de los laberintos a lo largo del tiempo", W3C Mayo 2013, Visitado el 23 Junio 2016** http://www.teinteresasaber.com/2013/05/origen-e-historia-de-los-laberintos-lo.html

[10] **"EL LABERINTO DE EGIPTO", W3C, Visitado el 10 de Junio de 2016** http://www.labolab.net/mitologia/el-laberinto-de-egipto/

[11] **"EL LABERINTO ROMANO", W3C Visitado el 11 de Junio de 2016** http://www.labolab.net/tipos/laberinto-unicursal/el-laberinto-romano/

[12] **"Laberinto de Creta", W3C 28 Mayo 2016, Visitado el 15 de Junio de 2016** https://es.wikipedia.org/wiki/Laberinto_de_Creta

[13] **"EL LABERINTO MEDIEVAL", Visitado el 17 de Junio de 2016** http://www.labolab.net/tipos/laberinto-unicursal/el-laberinto-medieval/

[14] **"Algoritmo para la generación de laberinto de conexión múltiple en 2D", W3C 2016, Visitado el 11 de Junio de 2016** http://www.academia.edu/3752288/Algoritmo_para_la_generaci%C3%B3n_de_laberintos_de_conexi%C3%B3n_m%C3%BAltiple_en_2D

[15] **"Maze Clasification", W3C 20 Noviembre 2015, Visitado el 17 de Junio de 2016** http://www.astrolog.org/labyrnth/algrithm.htm

[16] **"Arquitectura de los laberintos", W3C 2013, Visitado el 21 de Junio de 2016** https://issuu.com/inves/docs/arquitectura_de_los_laberintos-elena_garc_a-ies_in

[17] **"What´s a good algorithm to generate a maze?", W3C 1 Septiembre 2008, Visitado en Febrero de 2016** http://stackoverflow.com/questions/38502/whats-a-good-algorithm-to-generate-a-maze

[18] **"Maze Generation: Recursive Backtracking", W3C 27 Diciembre 2010, Visitado el 24 de Junio 2016** http://weblog.jamisbuck.org/2010/12/27/maze-generation-recursive-backtracking

[19] **"Maze Generation: Kruskal´s algorithm", W3C 3 Enero 2011, Visitado el 24 de Junio 2016** http://weblog.jamisbuck.org/2011/1/3/maze-generation-kruskal-s-algorithm

[20] **"Maze Generation Algorithm", W3C 4 Mayo 2016, Visitado el 20 de Febrero 2016**

https://en.wikipedia.org/wiki/Maze_generation_algorithm
[21] **"BREVE HISTORIA DEL LABERINTO CON SORPRESA FINAL", W3C 10 Octubre 2011, Visitado el 15 de Junio 2016.** http://tierradeamacos.blogspot.com.es/2011/10/breve-historia-del-laberinto-con.html
[22] **"Laberintos del Mundo: Historias mitológicas y leyendas", W3C 18 Mayo 2009, Visitado el 11 de Junio 2016** https://aikun.wordpress.com/2009/05/18/laberintos-del-mundo-historias-mitologicas-y-leyendas/
[23] **"Breve historia de los laberintos", W3C Julio 2014, Visitado el 12 de Junio 2016** http://arqui-2.blogspot.com.es/2014/07/breve-historia-de-los-laberintos.html
[24] **"Patrón de diseño", W3C 30 Mayo 2016** https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o
[25] **"Prototype (Patrón de diseño)", W3C 21 Abril 2016, Visitado el 19 de Junio 2016** https://es.wikipedia.org/wiki/Prototype_(patr%C3%B3n_de_dise%C3%B1o)
[26] **"Diccionario de la lengua española: Laberinto", W3C Visitado el 20 de Mayo 2016** http://dle.rae.es/?id=MihipGi

Glosario

API	Application Programming Interface.
Asset	Denominación que se le da a las librerías para Unity.
Bake	Proceso de creación del navigation automatizado de Unity.
Forma ovoidal	forma de huevo.
GameObject	Tipo de objeto genérico de Unity.
GridLayout	Tipo de objeto basado en una malla.
GPU	Graphics Processor Unit.
IA	Inteligencia Artificial.
Juego indie	Juego de desarrollo independiente.
Navigation	Mapa en Unity indicando el camino por el que las IAs se pueden mover.
NavMeshObstacle	Elemento perteneciente a un objeto de Unity que indica que ese objeto no puede ser atravesado por una IA y esta debe esquivarlo.
Pseudocódigo	Descripción en lenguaje de alto nivel del código.
RAM	Random Access Memory.
RPG	Role Paying Game.
Script	Referente en Unity al archivo que contiene el código.
Terrain	Tipo de objeto de Unity designado para el terreno.

Anexos

A Manual de instalación

Instalación del Asset

Existen 2 métodos de instalación de Assets en Unity, los cuales son muy sencillos de realizar:

- **Método 1**: simplemente consiste en una vez abierto el proyecto en el cual se va a trabajar, arrastrar el Asset a la carpeta en donde se desea instalar el Asset desde el Gestor de Ventanas de Windows.

  Este método de instalación tiene como punto negativo que se instalan todos los elementos del Asset, es decir, que si el usuario no desease incluir algún elemento de la librería por que ya tiene algo que hace la misma funcionalidad, como por ejemplo el script de Celda y solo desea el GestorDeLaberintos se instalarían ambos y luego debería eliminar manualmente los elementos que no desee, que en este caso sería el script de Celda.

- **Método 2**:
  - Tal y como se muestra en la Figura 9: pestaña Assets/Import Package/Custom Package …

  - Se selecciona la ubicación del Asset del tipo .unitypackage y pulsar Abrir tal y como se ve en la Figura 10.

  - Seleccionar en qué carpeta se instalará y cuáles son los archivos seleccionándolos y deseleccionándolos con el check, como se ve en la Figura 11. Pulsando All se seleccionan todos, None se deseleccionan todos, Cancel cancela la instalación e Import realiza la instalación.

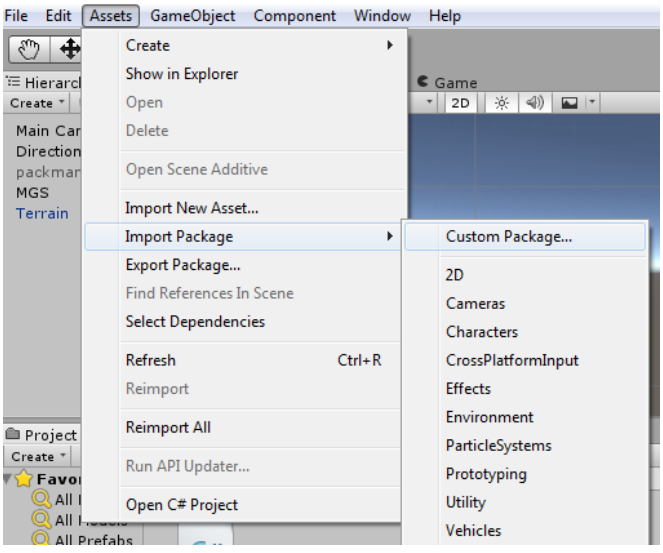


Figura 9: Instalación Asset método 2 (1)

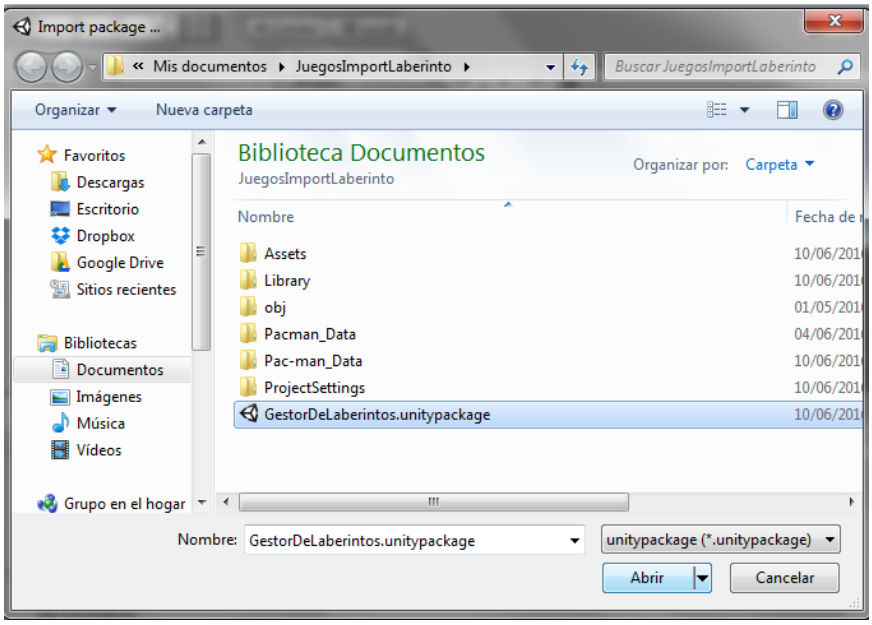


Figura 10: Instalación de Asset método 2 (2)

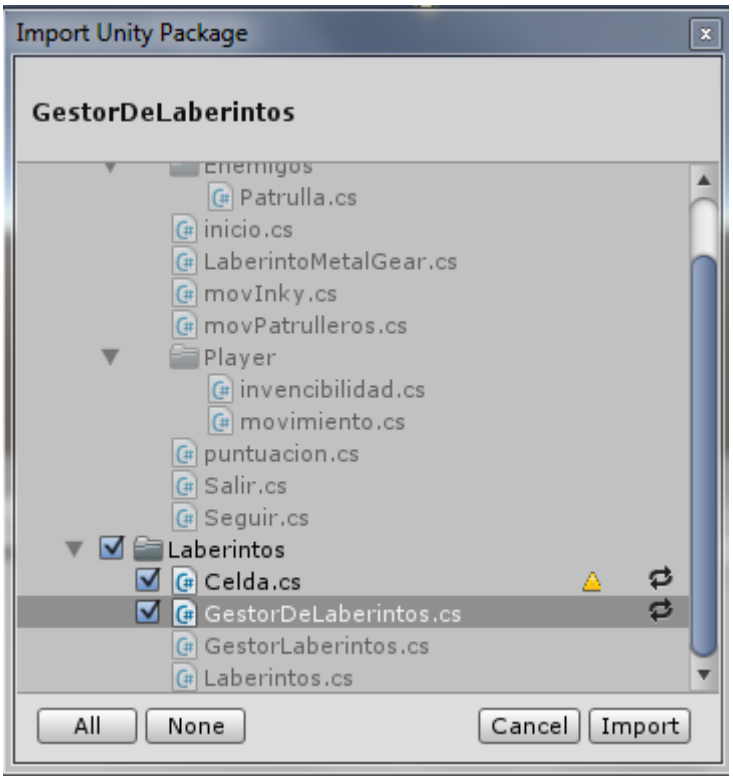


Figura 11: Instalación de Asset método 2 (3)

Instalación de Pacman

Descomprimir el archivo en la carpeta seleccionada y ya está instalado, para ejecutarlo simplemente basta con hacer doble clic en el archivo Pac-man.exe

B Manual del Asset

Para poder instanciar el objeto GestorDeLaberintos es necesario crearlo de la siguiente forma:

```
GestorDeLaberintos gdl = new
GameObject.AddComponent<GestorDeLaberintos>();
```

Una función que imita a los constructores clásicos para que el usuario no se sienta perdido es la siguiente:

gdl.GestorDeLaberintos();

Los posibles parámetros entre paréntesis, aparte de ir sin ninguno, son los siguientes, colocados en el orden indicado:

- `Int ancho, int largo`
- `Int ancho, int largo, GameObject muros, Terrain suelos, GameObject player`
- `Int ancho, int largo, GameObject muros, Terrain suelos, GameObject player, pjRand, intInt`

Para poder acceder a cualquiera de las variables públicas se utiliza getNombreVariable y para cambiar el valor de las variables se utiliza setNombreVariable(valor):

- `Float x = getTamMuroX;`
- `setTamMuroX(5.5f);`

Las variables públicas son: proEstadosRep, tamMuroX, tamMuroY, tamMuroZ, ancho, largo, muro, suelo, jugador, jugadorAleatorio, salaInicio.

Para crear un laberinto hay que emplear la función:
```
GameObject go= crearLaberinto(int tipo);
```

Donde int tipo es un entero entre 0 y 3 y retorna un GameObject que es el laberinto.

Para eliminar el laberinto del escenario basta con emplear la función:
```
borrarLaberinto();
```

Para colocar un jugador están las funciones siguientes:

- Para colocar un jugador dependiendo del tipo de inserción indicada en jugadorAleatorio, true es aleatorio, false no:
  ```
  GameObject pj = colocarJugador();
  ```
- Para colocar un jugador en una celda aleatoria:
  ```
  GameObject pj =  colocarJugadorRandomXSalsas();
  ```

- Para colocar un jugador en la celda especificada:
  ```
  GameObject pj = colocarJugadorSalaInicio();
  ```

- Para colocar un jugador en una celda aleatoria en un laberinto con varios pisos:
  ```
  GameObject pj =  colocarJugadorRandomXSalsas3D();
  ```

- Para colocar un jugador en la celda especificada en un laberinto con varios pisos:
  ```
  GameObject pj = colocarJugadorSalaInicio3D();
  ```

Para colocar GameObject dentro del laberinto están las siguientes funciones:

- Para colocar un GameObject en una celda especificada:
  ```
  GameObject ob = colocarGameObject(GameObject obj, int sala);
  ```

- Para colocar un GameObject en una celda aleatoria:
  ```
  GameObject ob = colocarGameObjectXSalas(GameObject obj);
  ```

- Para colocar un GameObject en una celda especificada en un laberinto con varios pisos:
  ```
  GameObject ob = colocarGameObject3D(GameObject obj, int sala, int piso);
  ```

- Para colocar un GameObject en una celda aleatoria en un laberinto con varios pisos:
  ```
  GameObject ob = colocarGameObjectXSalas3D(GameObject obj);
  ```

Para mover GameObject dentro del laberinto están las siguientes funciones:

- Para colocar un GameObject en una celda especificada:
  ```
  moverGameObject(GameObject obj, int sala);
  ```

- Para mover un GameObject en una celda aleatoria:
  ```
  moverGameObjectXSalas(GameObject obj);
  ```

- Para colocar un GameObject en una celda especificada en un laberinto con varios pisos:
  ```
  moverGameObject3D(GameObject obj, int sala, int piso);
  ```

- Para mover un GameObject en una celda aleatoria en un laberinto con varios pisos:
  ```
  moverGameObjectXSalas3D(GameObject obj);
  ```

Para obtener una posición aleatoria de una celda dentro del laberinto está la siguiente función:

```
Vector3 v= getPosRandXSalas();
```

Para generar el mapa de navegación de los muros del laberinto está la función:

```
FixNavMesh();
```

C Anexo Pacman

Requisitos funcionales

Requisitos funcionales y características referentes a la implementación del Asset del generador de laberintos en un juego del Pacman.

Código	Requisito
FP-01	Importar el Asset del generador de laberintos, el mapa son los laberintos generados por el Asset.
FP-02	Escena de inicio con opciones de jugar o salir.
FP-03	Sistema de puntuaciones obtenidas al comer un fantasma o al comer una esfera.
FP-04	Sistema de vidas, inicialmente 3. Si no hay vidas se pierde.
FP-05	Sistema de enemigos con diversas personalidades: - Fantasma rojo: persigue a Pacman de manera agresiva con búsqueda en A* - Fantasma rosa: persigue a Pacman de manera menos agresiva con búsqueda en A* - Fantasma naranja: movimiento libre - Fantasma azul: intenta evitar a Pacman.
FP-06	El jugador puede moverse libremente con Pacman dentro del laberinto.
FP-07	Al comer una Bola rosa los fantasmas huyen de Pacman durante 15 segundos y cambian su textura al del fantasma azul, después vuelven a la normalidad.
FP-08	Al perder las vidas o comer todas las esferas finaliza el juego.
FP-09	El jugador puede reiniciar el juego o salir una ver que este ha finalizado.
FP-10	Los laberintos son con multiconexiones, ya que las otras generaciones de laberintos impedirían la jugabilidad al existir un único camino pues los fantasmas siempre alcanzan al jugador.
FP-11	Los fantasmas empiezan cada uno en las casillas centrales
FP-12	Pacman empieza en una casilla aleatoria para darle innovación al juego.

Tabla 5: Requisitos funcionales de Pacman.

Requisitos no funcionales

Requisitos no funcionales de la aplicación del Pacman el cual implementa el Asset del generador de laberintos.

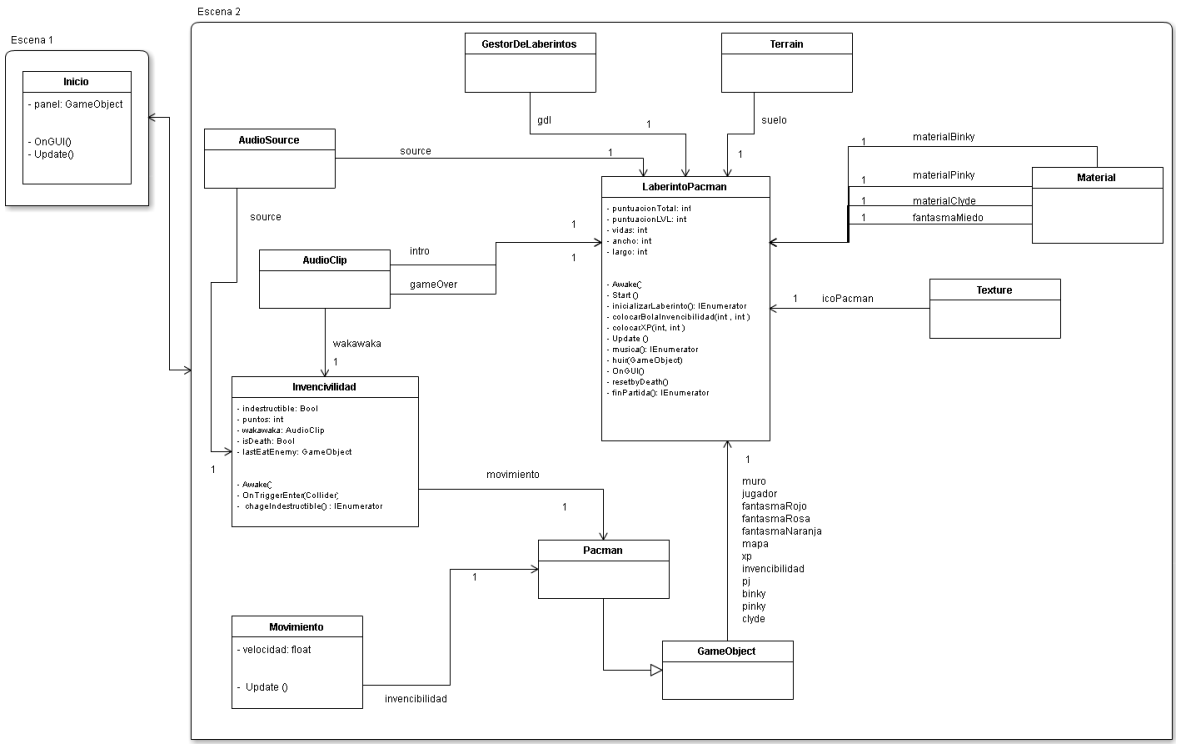
Código	Requisito
NFP-01	Usabilidad: Interfaz Sencilla e intuitiva, se maneja a Pacman con las flechas de dirección.
NFP-02	Rendimiento: Juego con unos requisitos mínimos no muy elevados.

NFP-03	Portabilidad: No requiere de instalaciones, solo de la carpeta con el .exe y las dependencias.

Tabla 6: Requisitos no funcionales de Pacman

Diagrama de clases

La muestra el diagrama de clases de la implementación del Asset dentro del juego del Pacman.



Estructura de los objetos en las carpetas:

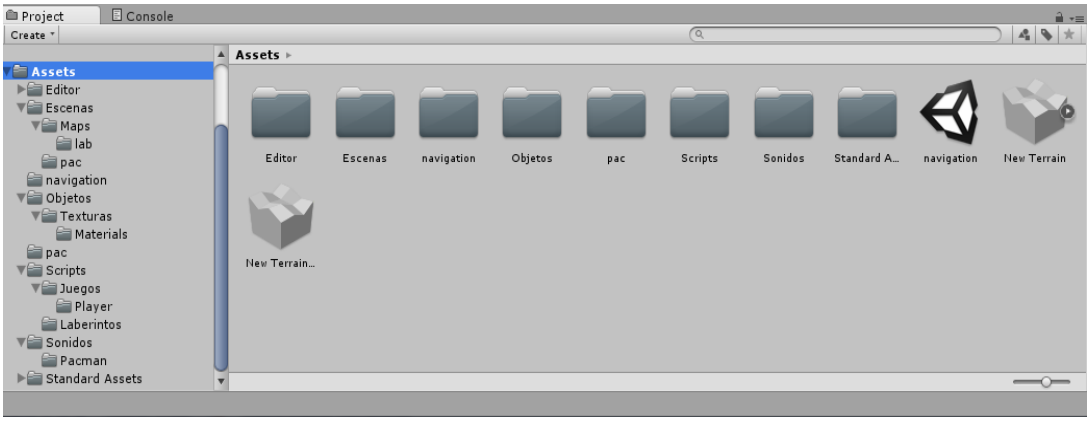


Figura 12: Carpetas de Pacman.

Tal y como se ve en la Figura 12, la implementación posee varias carpetas con distintas subcarpetas, las más importantes son:

- **Escenas:** En esta carpeta se hallan principalmente las escenas InicioPacman y Pac, los cuales corresponden a la pantalla de inicio y al juego respectivamente, como se ve en la Figura 13.

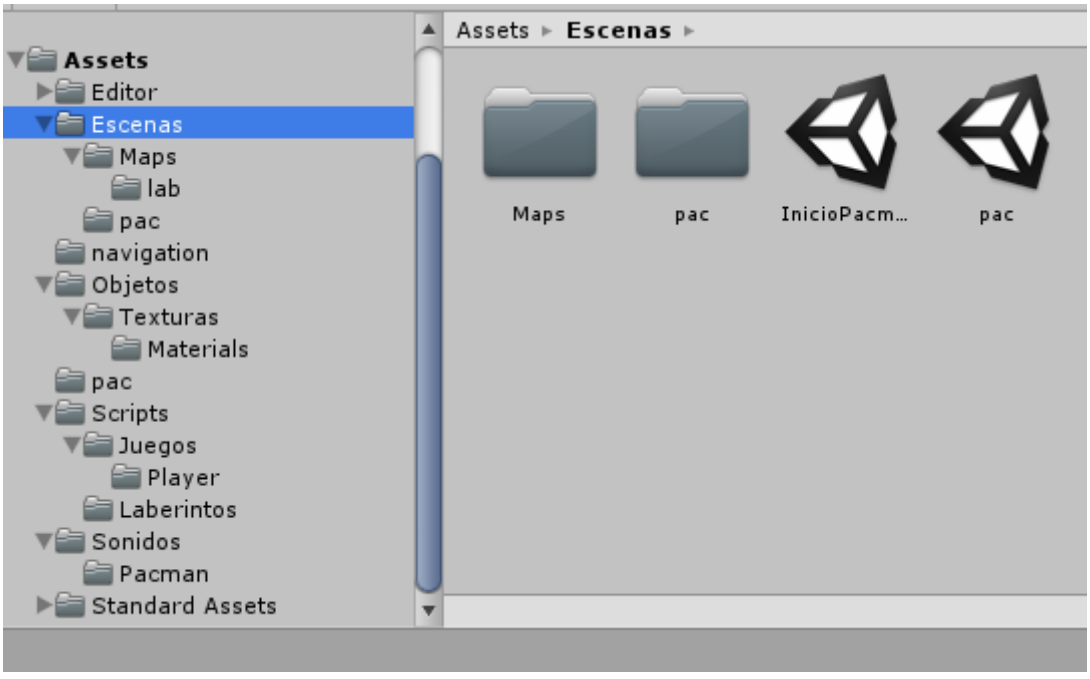


Figura 13: Elementos de la carpeta Escenas.

- **Objetos:** En esta carpeta se hallan todos los objetos del juego, considerados como objetos genéricos ya que si se modifican éstos, todos los que ya existieran insertados también cambian, estos se ven en la Figura 14.

Posee una carpeta Texturas la cual posee todas las texturas pertenecientes al juego, como se puede ver en la Figura 15. Esta carpeta también posee una subcarpeta Materials en donde se encuentran los materiales empleados en el juego. Estos se ven en la Figura 16.

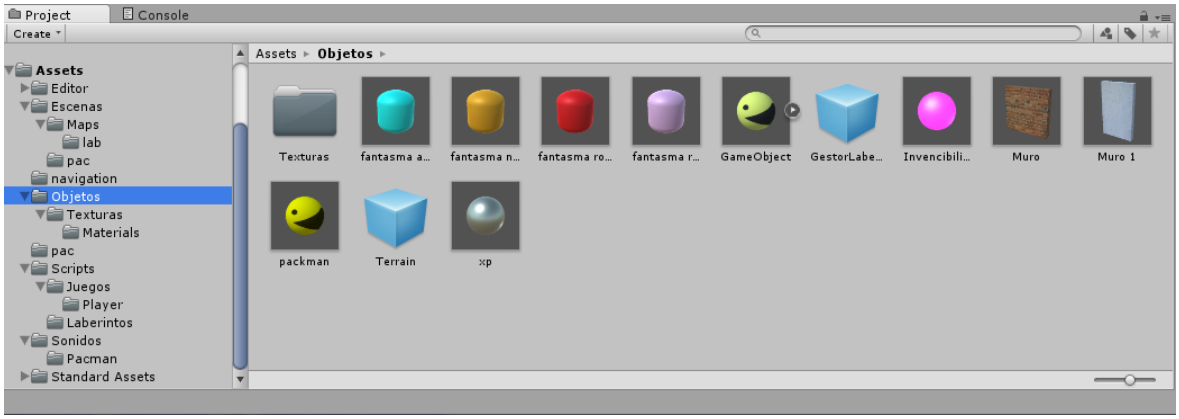


Figura 14: Elementos de la carpeta Objetos.

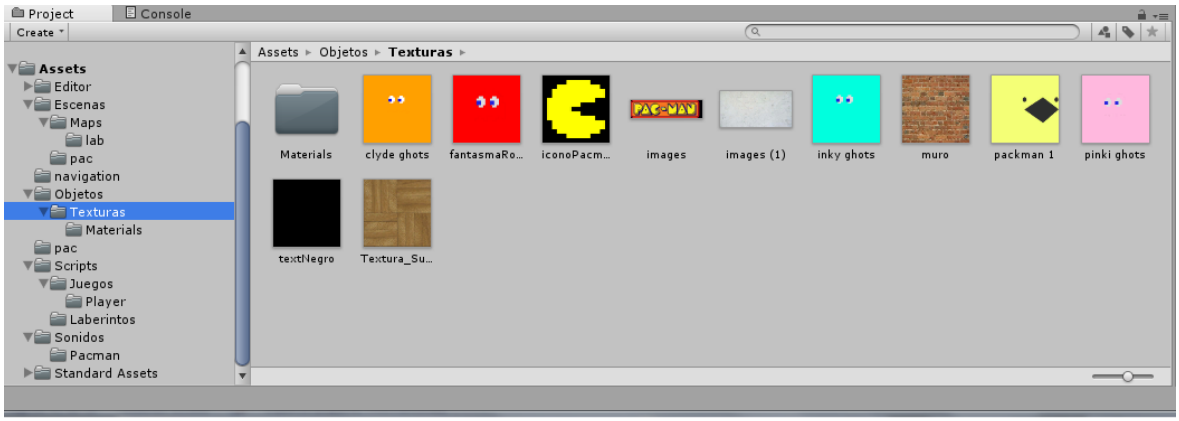


Figura 15: Elementos de la subcarpeta Texturas

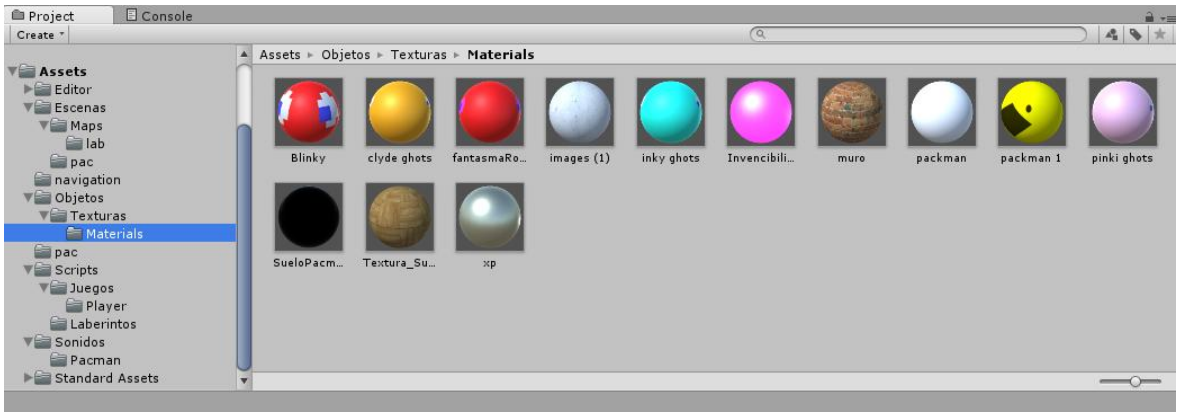


Figura 16: Elementos de la subcarpeta Materials.

- **Scripts:** En esta carpeta se encuentran los scripts del juego. Se divide en dos subcarpetas: Laberintos y Juegos:

    - **Laberintos:** en esta carpeta se ubican los scripts de Celda.cs y GestorDeLaberintos.cs.

    - **Juegos:** En esta carpeta se ubican los scripts relacionados con el juego del Pacman. Se encuentran en el raíz los scripts relacionados con los escenarios: Inicio, LaberintoPacman, puntuación y salir, como se ve en la Figura 17 y por otro lado, está la carpeta Player, que contiene los scripts del jugador, como se ve en la Figura 18.

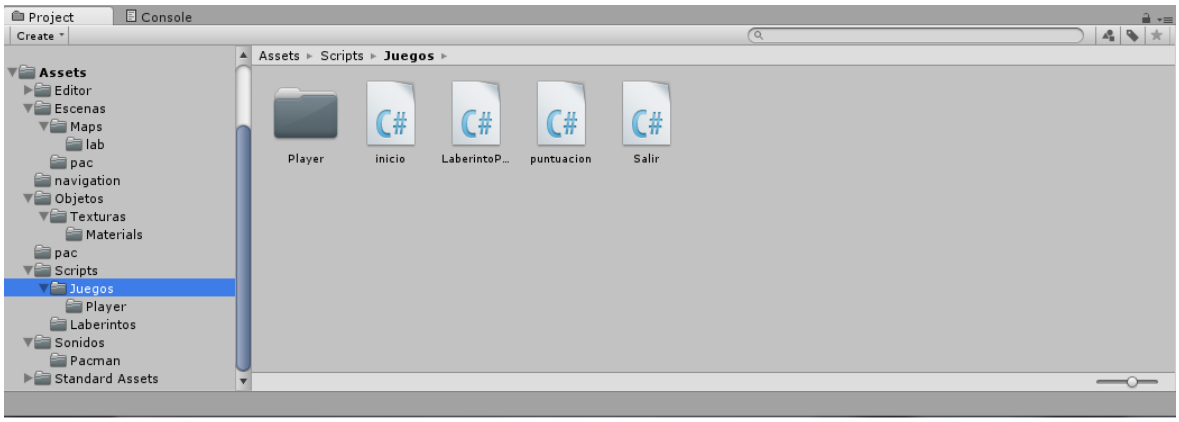


Figura 17: Elementos de la subcarpeta Juegos

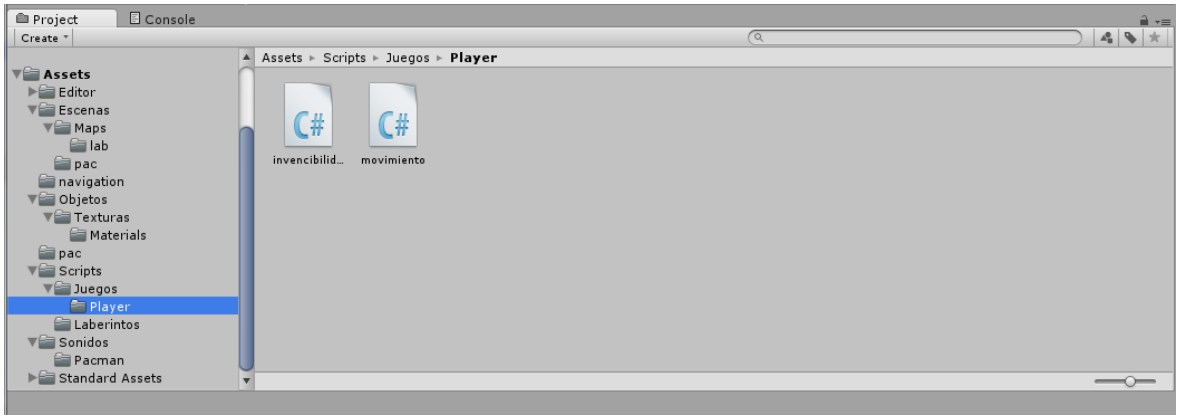


Figura 18: Elementos de la subcarpeta Player

- **Sonidos:** En esta carpeta se encuentran los archivos de sonido del juego del Pacman como se ve en la Figura 19, aunque estén ubicados en la subcarpeta Pacman.

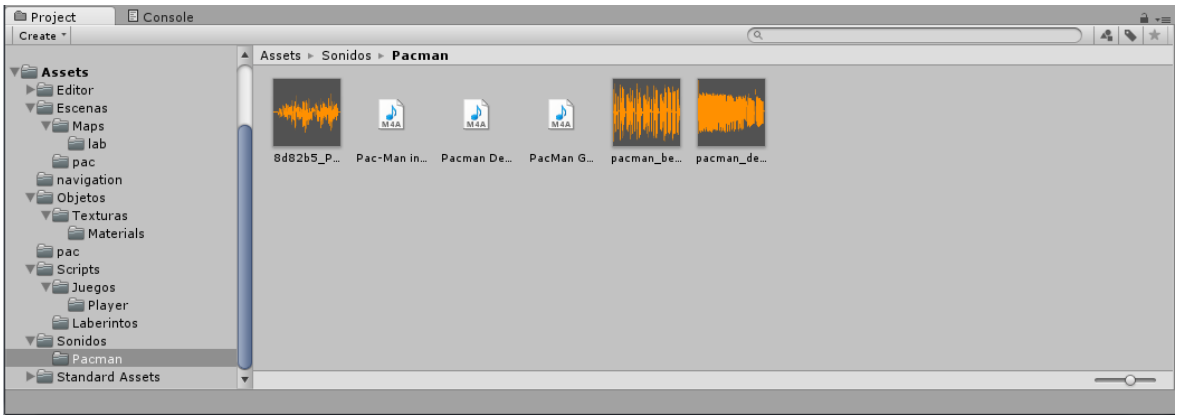


Figura 19: Elementos de la Carpeta Sonidos/Pacman

D Anexo Imágenes de los test

Debido a que los test realizados son puramente gráficos, no se han podido generar ningún informe de entradas y salidas esperadas salvo indicar qué se obtendría de las ejecuciones de las implementaciones, las cuales son los test.

Estas son las imágenes de las pruebas realizadas sobre el generador para cada una de las implementaciones:

- **Laberintos:**

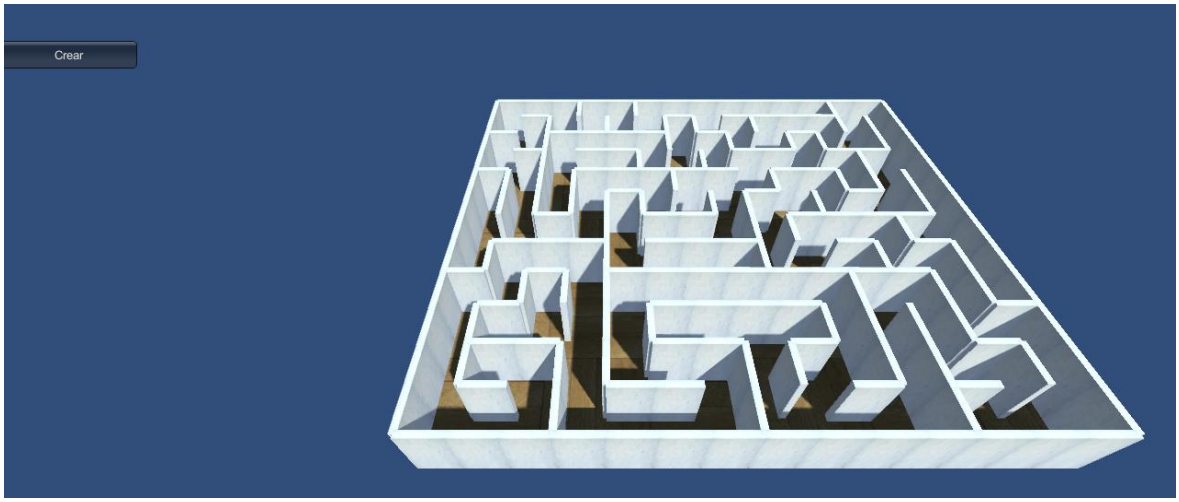


Figura 20: Generación algoritmo búsqueda en profundidad.

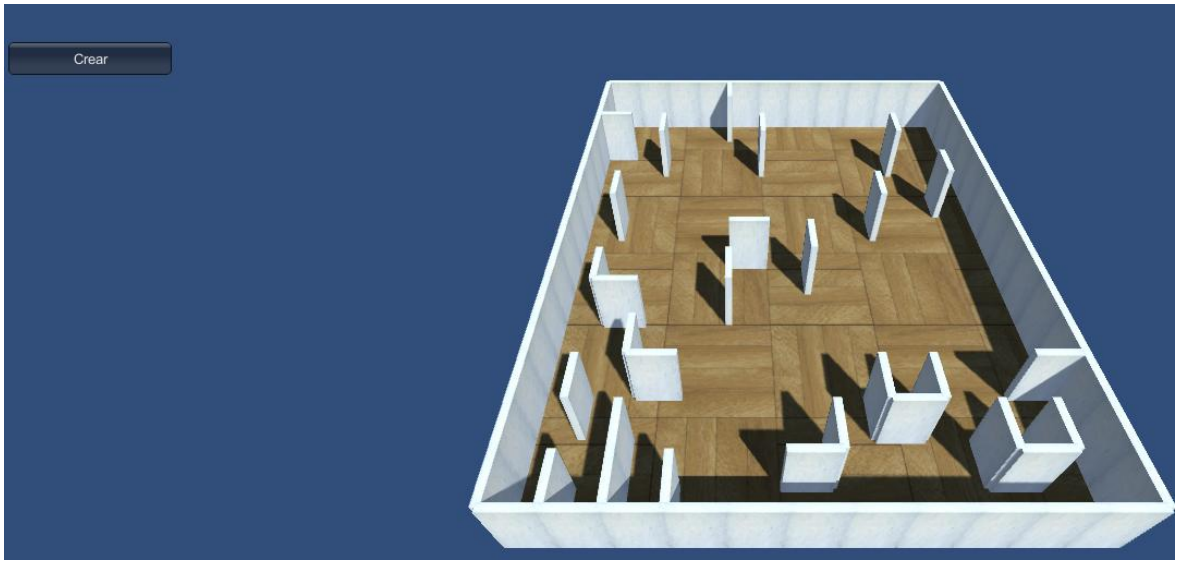


Figura 21: Generación algoritmo búsqueda en profundidad con multiconexión.

Figura 22: Generación algoritmo Árbol binario.

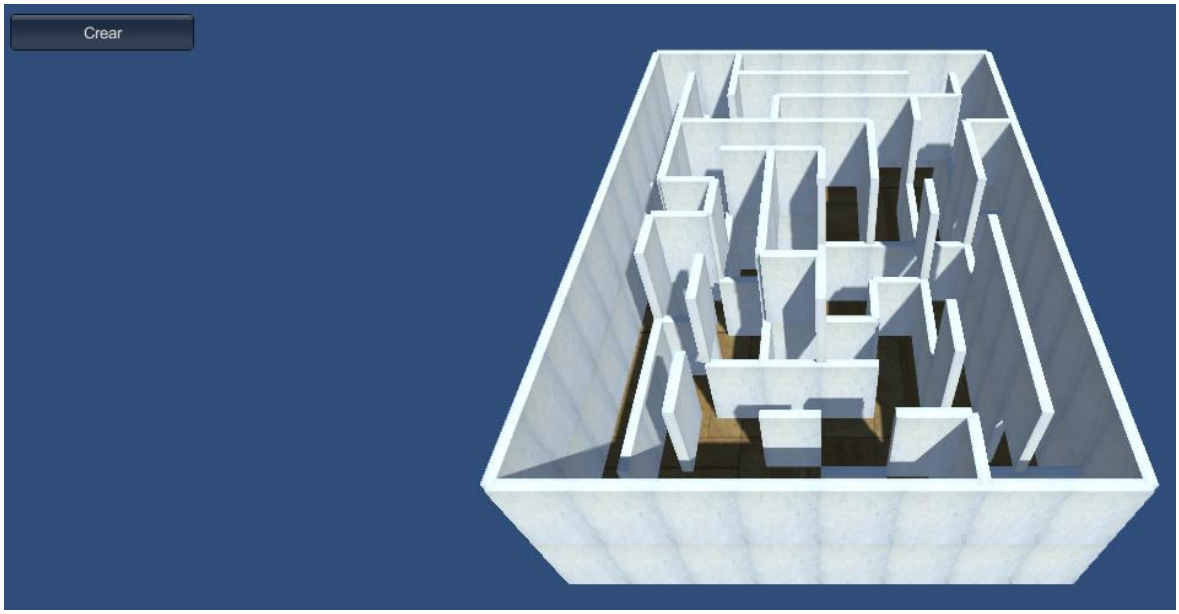


Figura 23: Generación multiconexión de varios pisos.

Figura 24: Generación algoritmo Árbol binario incompleto.

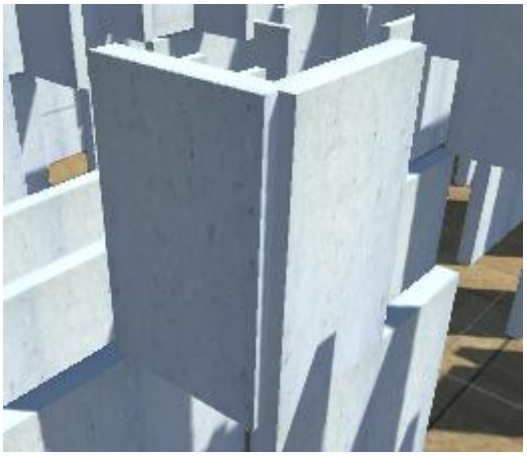


Figura 25: Colocación correcta de los muros en los pisos.

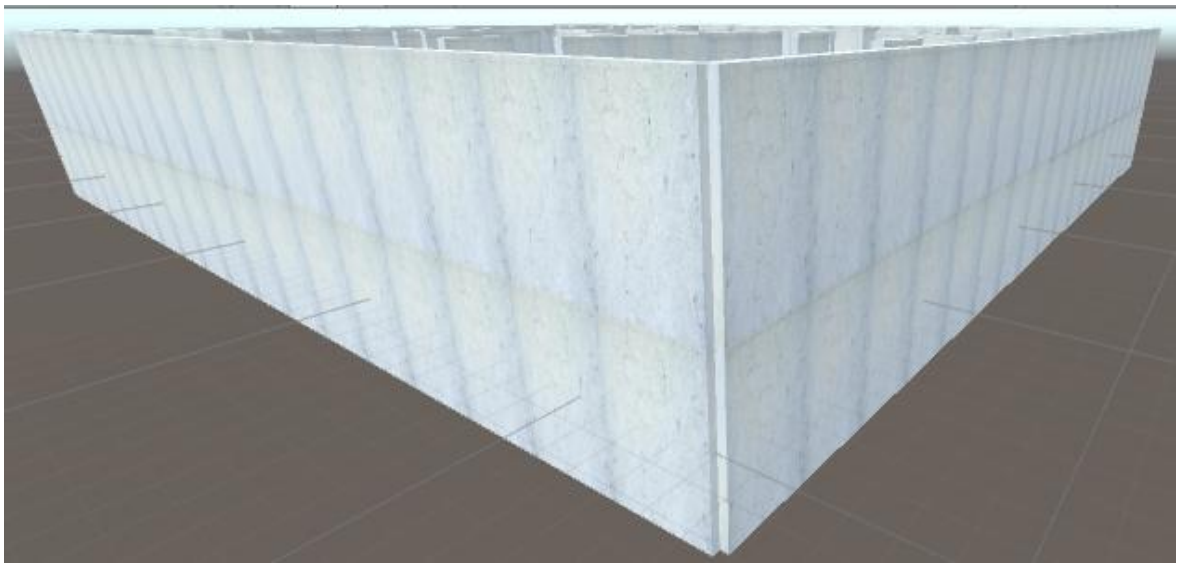


Figura 26: Muros exteriores completos.

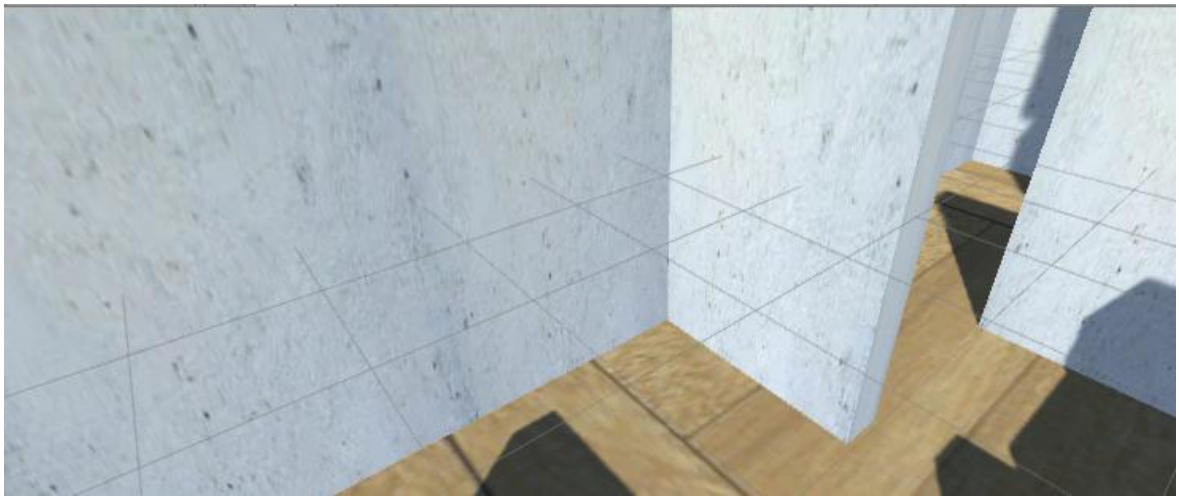


Figura 27: Colocación correcta del suelo.

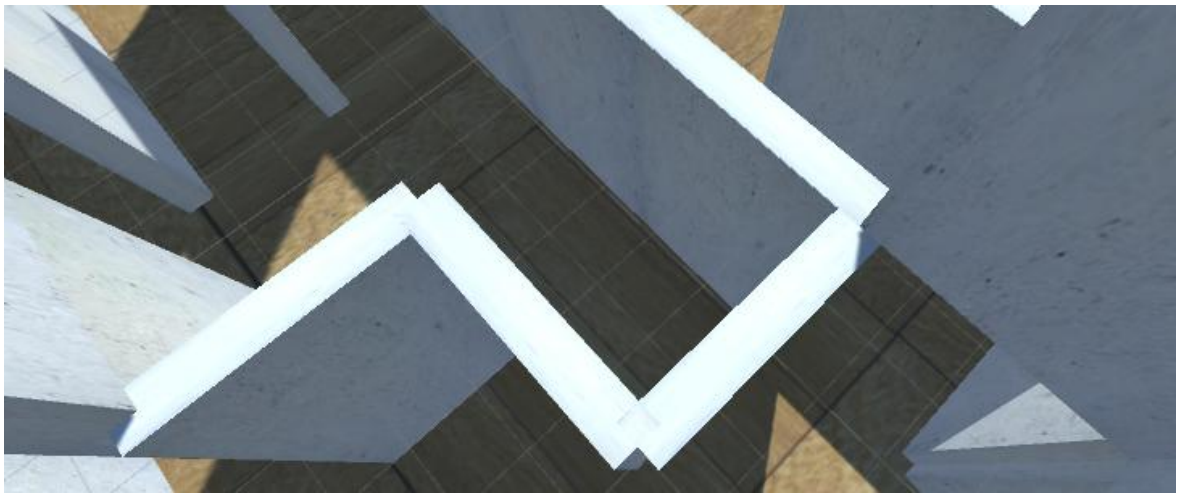


Figura 28: Colocación de los muros formando las esquinas sin huecos.

Figura 29: Laberinto con 3 de alto.

- **Patrulleros:**

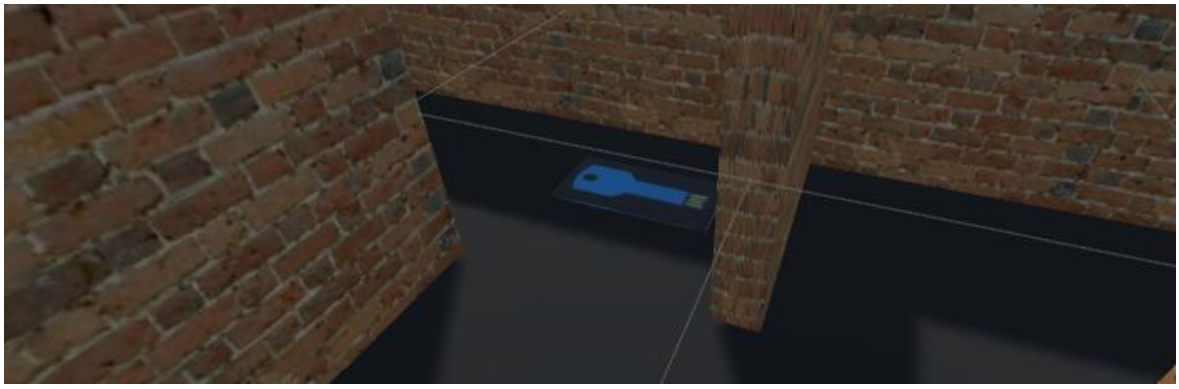


Figura 30: Colocación correcta de los GameObject.

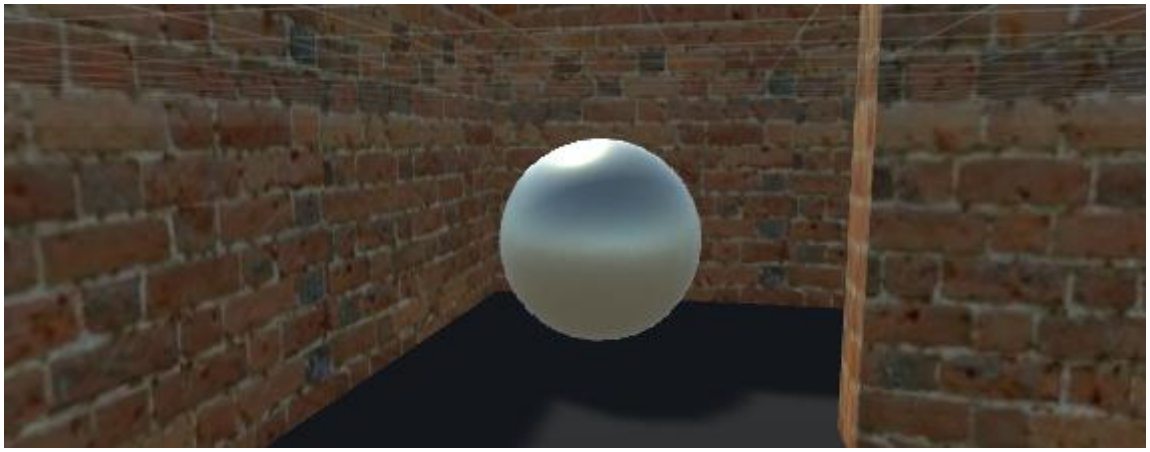


Figura 31: Colocación correcta de los GameObject.

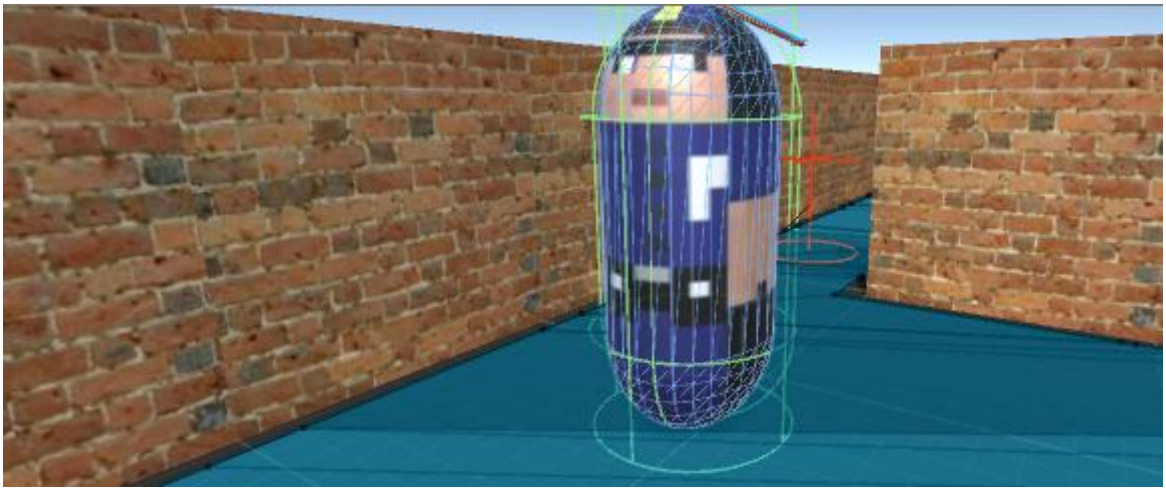


Figura 32: IAs con navigation

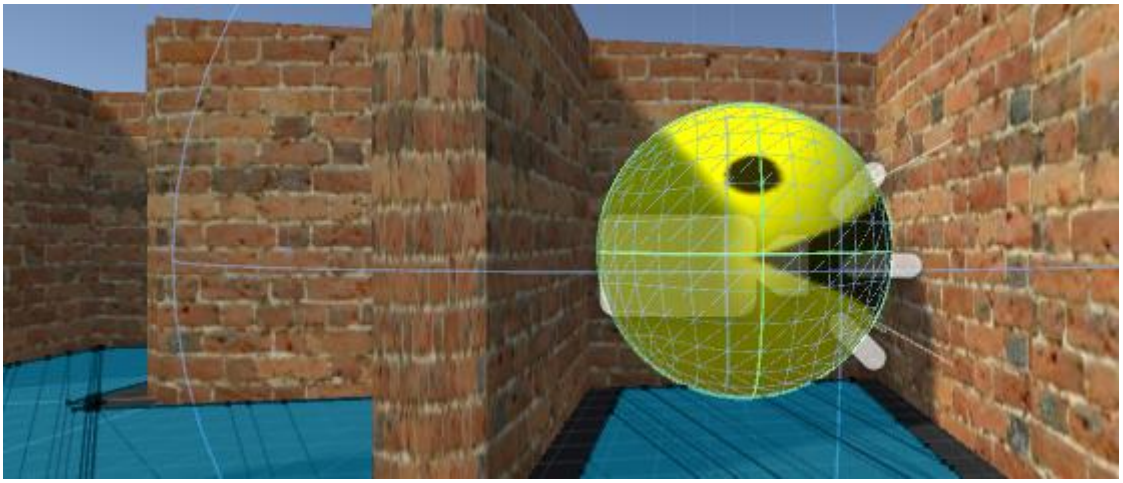


Figura 33: Colocación correcta del jugador.

Figura 34: Mapa de navigation correcto, sin agujeros.

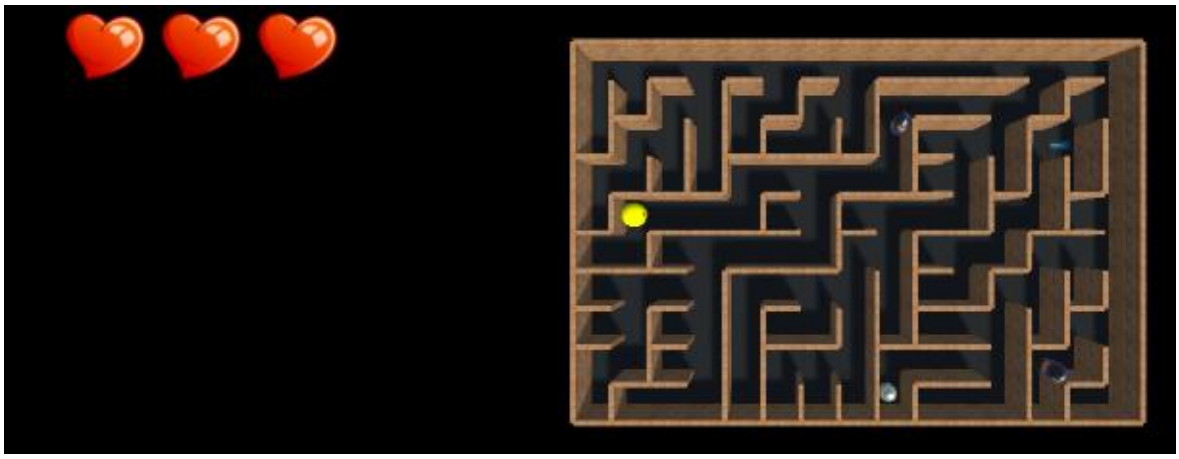


Figura 35: Colocación aleatoria.

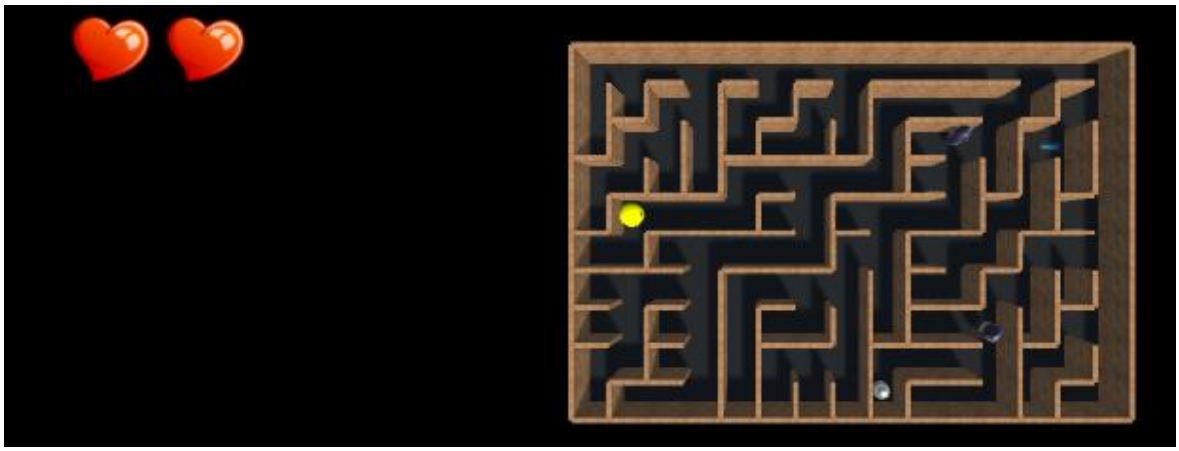


Figura 36: Movimiento de las IAs con respecto a la figura anterior.

Figura 37: Movimiento del jugador por el laberinto.

- **Pacman:**

Figura 38: Escena de la introducción.

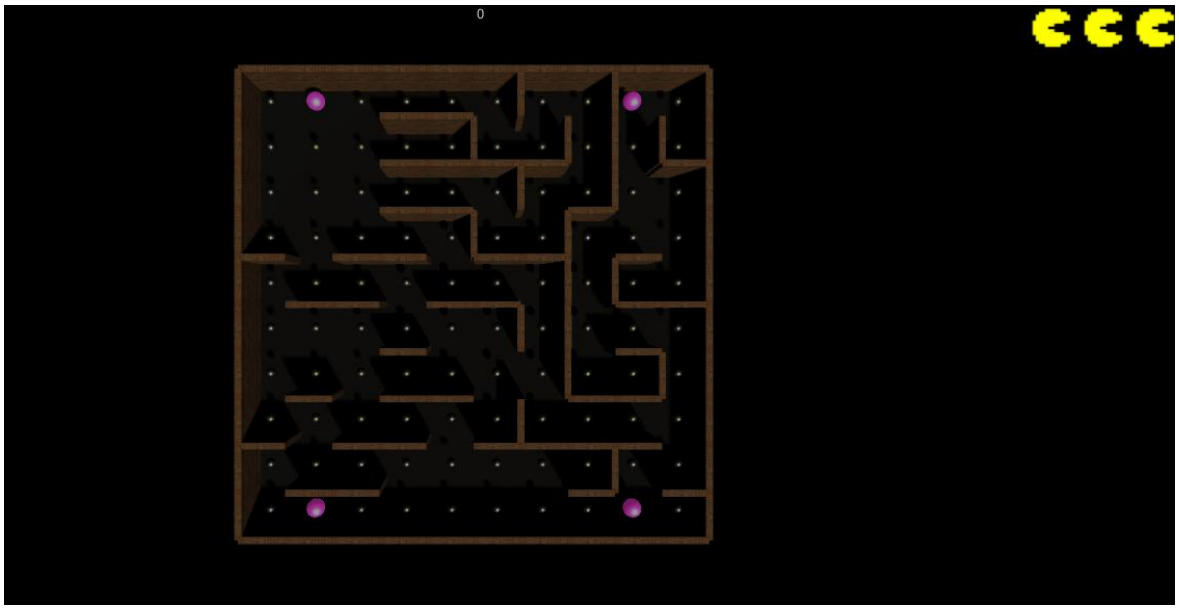


Figura 39: Inserción de los GameObject.

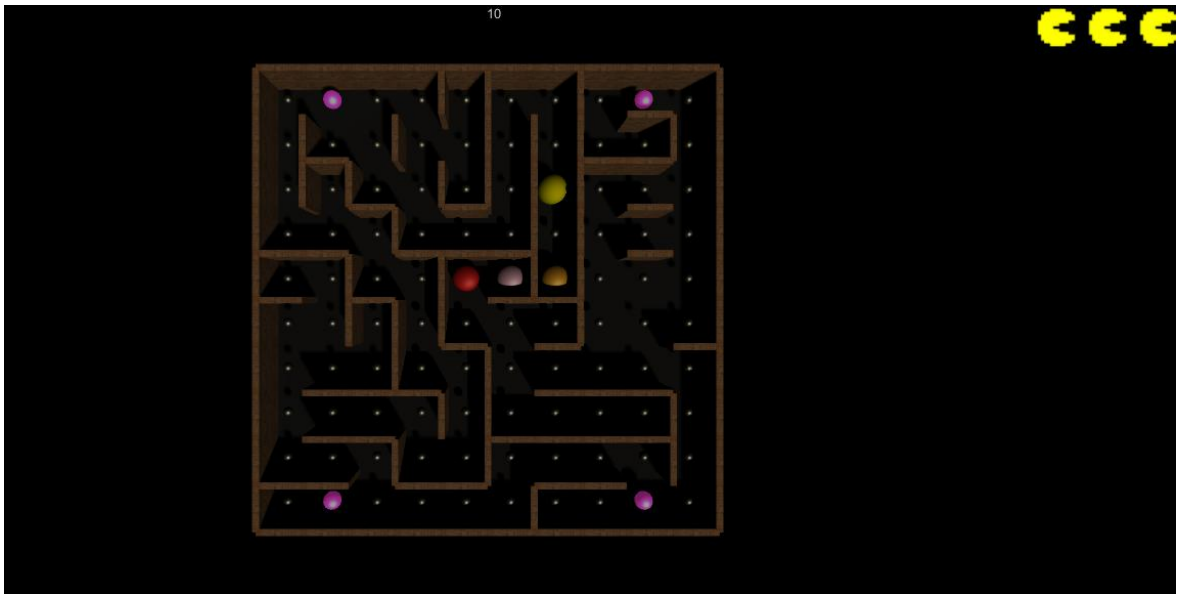


Figura 40: Colocación de los fantasmas en la posición una posición única.

Figura 41: Fantasmas en modo huida de Pacman.

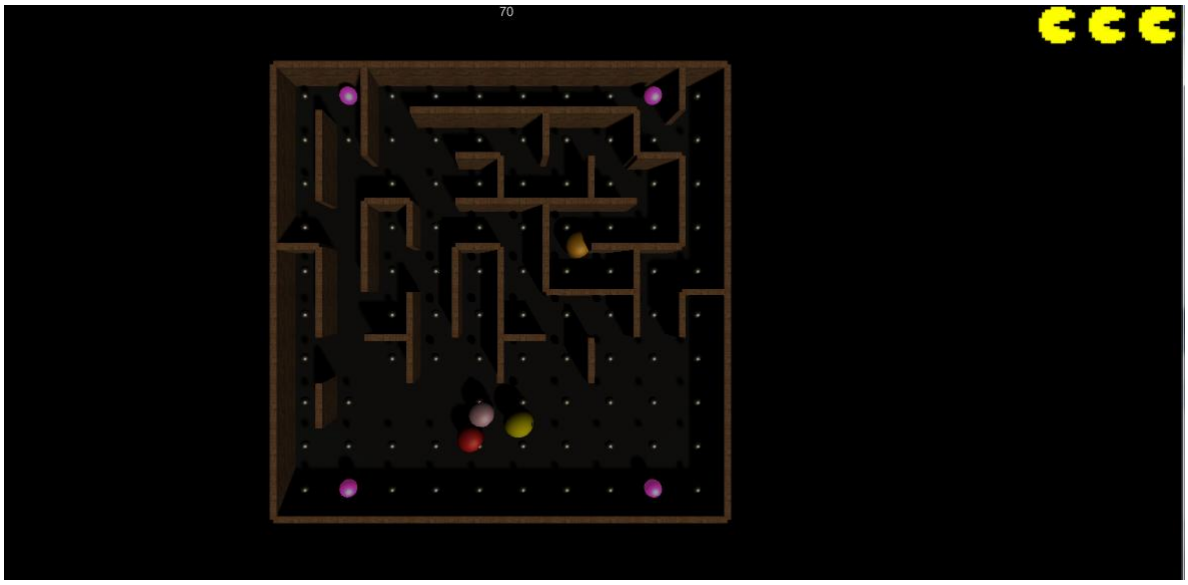


Figura 42: Fantasmas persiguiendo a Pacman según su algoritmo.

E Anexo Guía de las implementaciones.

En este anexo se explicará cómo utilizar las implementaciones.

Laberintos:

- Una vez descargado .rar desde el link, se le hace doble clic al archivo Laberintos.exe.
- Para generar laberintos solo es necesario pulsar el botón Crear en la parte superior izquierda de la pantalla.
- Para salir pulsar la barra espaciadora o si se está en modo ventana cerrar la ventana de Windows.

Link:
https://www.dropbox.com/s/bu1hkj36ioko2u0/Laberintos.rar?dl=0

Patrulleros:

- Una vez descargado .rar desde el link, se le hace doble clic al archivo Patrulleros.exe.
- Para moverse por el laberinto hay que usar las flechas de movimiento.
- Para ganar tienes que coger la llave y después la esfera.

- Pierdes si te atrapan 3 veces los policías y te quitan los 3 corazones.
- Para salir del juego pulsar la barra espaciadora.
- Para generar reiniciar el juego con un nuevo laberinto pulsar r.
- Al ganar o perder te da la opción de salir o volver a jugar.

Link:
https://www.dropbox.com/s/uyadisvc4gy17cv/Patrulleros.rar?dl=0

Pacman:

- Una vez descargado el .rar desde el link, se le hace doble clic al archivo Pac-man.exe.
- Se empieza desde la pantalla de inicio. Para jugar pulsar jugar, para salir pulsar salir.
- Para moverse por el laberinto hay que usar las flechas de movimiento.
- Para completar el juego hay que obtener todas las esferas del laberinto.
- Pierdes si te atrapan 3 veces los fantasmas y te quitan todas las vidas.
- Atrapar fantasmas azules aumenta la puntuación.
- Para salir del juego pulsar la barra espaciadora.
- Para generar reiniciar el juego con un nuevo laberinto pulsar r.
- Al ganar o perder te da la opción de salir o volver a jugar.

Link:
https://www.dropbox.com/s/29i1d0of9jj92jk/Pacman.rar?dl=0

Link del Asset:
https://www.dropbox.com/s/8pq0cvw3fuyvkx9/Asset.rar?dl=0